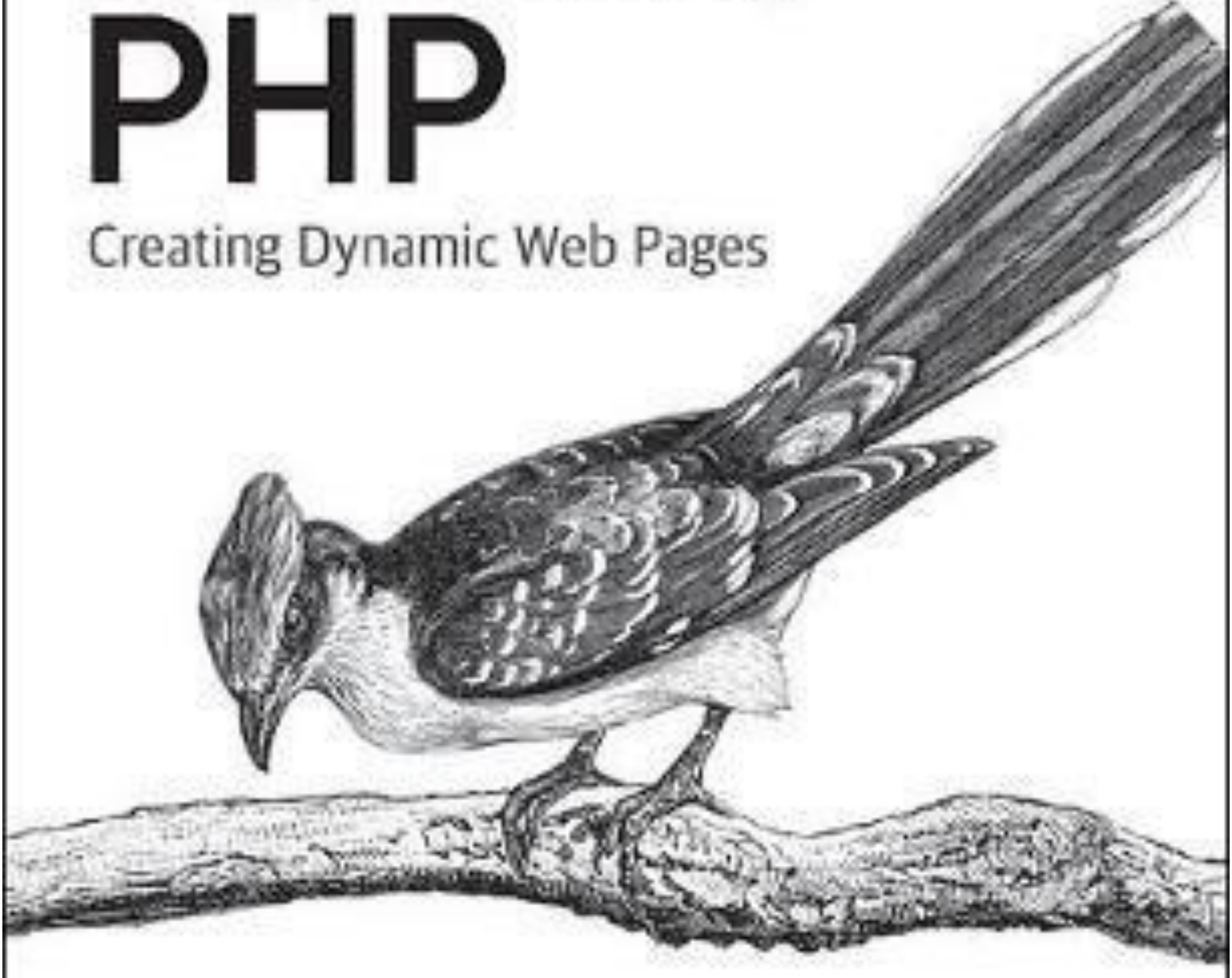
O'REILLY

Fourth Edition

Programming PHP

Creating Dynamic Web Pages



Kevin Tatroe &
Peter MacIntyre

الفصل الأول مقدمة لـ PHP

PHP هي لغة بسيطة لكنها قوية مصممة لإنشاء محتوى HTML. يغطي هذا الفصل الخلفية الأساسية للغة PHP. يصف طبيعة وتاريخ PHP، والأنظمة الأساسية التي يعمل عليها، وكيفية تكوينها. ينتهي هذا الفصل بعرض PHP عمليًا، مع عرض سريع للعديد من برامج PHP التي توضح المهام الشائعة، مثل معالجة بيانات النموذج، والتفاعل مع قاعدة بيانات، وإنشاء رسومات.

ماذا تفعل PHP؟

يمكن استخدام PHP بطريقتين أساسيتين:

البرمجة النصية من جانب الخادم

تم تصميم PHP في الأصل لإنشاء محتوى ويب حيوي، ولا يزال ملائمًا لهذه المهمة. لتوليد HTML، تحتاج إلى محلل PHP وخادم ويب لإرسال ملفات المستندات المشفرة. أصبح PHP أيضًا شائعًا في إنشاء محتوى حيوي عبر اتصالات قاعدة البيانات ومستندات XML والرسومات وملفات PDF والمزيد.

برمجة سطر الأوامر

يمكن لـ PHP تشغيل نصوص برمجية من سطر الأوامر، تمامًا مثل Perl أو awk أو shell Unix. يمكنك استخدام البرامج النصية لسطر الأوامر لمهام إدارة النظام، مثل النسخ الاحتياطي وتحليل السجلات؛ حتى بعض البرامج النصية من نوع الوظيفة CRON يمكن القيام بها بهذه الطريقة (كمهام PHP غير مرئية).

في هذا الكتاب، نركز على العنصر الأول: استخدام PHP لتطوير محتوى ويب حيوي.

تعمل PHP على جميع أنظمة التشغيل الرئيسية، من متغيرات Unix (بما في ذلك Linux و FreeBSD و Ubuntu و Debian و Solaris) إلى Windows و macOS. يمكن استخدامه مع جميع خوادم الويب المعروفة، بما في ذلك خوادم Apache و Nginx و OpenBSD، على سبيل المثال لا الحصر؛ حتى البيئات السحابية مثل Azure و Amazon في ازدياد.

اللغة نفسها مرنة للغاية. على سبيل المثال، لا تقتصر على إخراج HTML أو ملفات نصية أخرى فقط - يمكن إنشاء أي تنسيق مستند. يحتوي PHP على دعم مدمج لتوليد ملفات PDF وصور GIF و JPEG و PNG.

واحدة من أهم ميزات PHP هو دعمها الواسع لقواعد البيانات. يدعم PHP جميع قواعد البيانات الرئيسية (بما في ذلك MySQL و PostgreSQL و Oracle و Sybase و MS-SQL و DB2 وقواعد البيانات المتوافقة مع ODBC)، وحتى العديد من قواعد البيانات الغامضة. حتى قواعد البيانات الأحدث مثل NoSQL مثل CouchDB و MongoDB مدعومة أيضًا. باستخدام PHP، يعد إنشاء صفحات الويب ذات المحتوى الحيوي من قاعدة البيانات أمرًا بسيطًا للغاية.

وأخيرًا، توفر PHP مكتبة من كود PHP لأداء المهام الشائعة، مثل تجريد قاعدة البيانات ومعالجة الأخطاء وما إلى ذلك، باستخدام ملحق PHP وملف التطبيق (PEAR). PEAR هو إطار عمل ونظام توزيع لمكونات PHP قابلة لإعادة الاستخدام.

نبذة تاريخية عن PHP

تم تصميم Rasmus Lerdorf لأول مرة من PHP في عام 1994، ولكن PHP الذي يستخدمه الناس اليوم يختلف تمامًا عن الإصدار الأولي. لفهم كيف وصلت PHP إلى حيث هي الآن، من المفيد معرفة التطور التاريخي للغة. إليكم هذه القصة بتعليقات وافرة ورسائل بريد إلكتروني من راسموس نفسه.

تطور PHP

هنا هو إعلان PHP 1.0 الذي تم نشره في مجموعة أخبار Usenet comp.infosystems.www.authoring.cgi)) في يونيو 1995:

الموضوع: إعلان: أدوات الصفحة الرئيسية الشخصية "Personal Home Page Tools" (أدوات PHP)

التاريخ: 08/06/1995

مجموعات أخبار: comp.infosystems.www.authoring.cgi

هذه الأدوات هي مجموعة من ثنائيات cgi الصغيرة الضيقة المكتوبة في C.

يؤدون عددا من الوظائف بما في ذلك:

- تسجيل الدخول يصل إلى صفحاتك في ملفات السجل الخاصة بك
- عرض معلومات السجل في الوقت الحقيقي
- توفير واجهة جميلة لمعلومات السجل هذه

- عرض معلومات الوصول الأخيرة مباشرة على صفحاتك
- عدادات وصول يومية وإجمالية كاملة
- حظر الوصول إلى المستخدمين على أساس المجال الخاص بهم
- حماية كلمة المرور للصفحات بناء على مجالات المستخدمين
- تتبع الوصول ** بناءً على عناوين البريد الإلكتروني للمستخدمين **
- تتبع عناوين URL المرجعية - دعم HTTP_REFERER
- يتضمن أداء جانب الخادم دون الحاجة إلى دعم الخادم لذلك
- القدرة على عدم تسجيل الوصول من مجالات معينة (أي الخاصة بك)
- إنشاء النماذج وعرضها بسهولة
- القدرة على استخدام معلومات النموذج في المستندات التالية

إليك ما لا تحتاجه لاستخدام هذه الأدوات:

- لا تحتاج إلى الوصول إلى الجذر - قم بالتثبيت في ~ / public_html dir
- لا تحتاج إلى تمكين جانب الخادم في الخادم الخاص بك
- لا تحتاج إلى الوصول إلى Perl أو Tcl أو أي مترجم نص برمجي آخر
- لا تحتاج إلى الوصول إلى ملفات سجل httpd

الشرط الوحيد لهذه الأدوات للعمل هو أن لديك

القدرة على تنفيذ برامج cgi الخاصة بك. اسأل نظامك
المسؤول إذا كنت غير متأكد مما يعنيه هذا.

تسمح لك الأدوات أيضًا بتنفيذ دفتر الزوار أو أي كتاب آخر
النموذج الذي يحتاج إلى كتابة المعلومات وعرضها للمستخدمين
لاحقًا خلال دقيقتين تقريبًا.

الأدوات موجودة في المجال العام موزعة تحت جنو
رخصة عامة. نعم ، هذا يعني أنهم أحرار!

للحصول على عرض توضيحي كامل لهذه الأدوات ، قم بتوجيه المتصفح
في: http://www.io.org/~rasmus

-

راسموس ليردورف

rasmus@io.org

http://www.io.org/~rasmus

...

تثبيت PHP

كما ذكرنا، PHP متاح للعديد من أنظمة التشغيل والأنظمة الأساسية. لذلك، نشجعك على الرجوع إلى وثائق PHP للعثور على البيئة الأكثر ملاءمة للبيئة التي ستستخدمها واتباع تعليمات الإعداد المناسبة.

من وقت لآخر، قد ترغب أيضًا في تغيير طريقة تكوين PHP. للقيام بذلك، سيكون عليك تغيير ملف تكوين PHP وإعادة تشغيل خادم الويب (Apache). في كل مرة تقوم فيها بإجراء تغيير على بيئة PHP، سيتعين عليك إعادة تشغيل خادم الويب (Apache) حتى تدخل هذه التغييرات حيز التنفيذ.

عادةً ما يتم الاحتفاظ بإعدادات تكوين PHP في ملف يسمى php.ini. تتحكم الإعدادات الموجودة في هذا الملف في سلوك ميزات PHP، مثل معالجة الجلسة ومعالجة النماذج. تشير الفصول اللاحقة إلى بعض خيارات php.ini، لكن بشكل عام لا تتطلب الكود في هذا الكتاب تهيئة مخصصة. راجع وثائق PHP لمزيد من المعلومات حول تكوين php.ini.

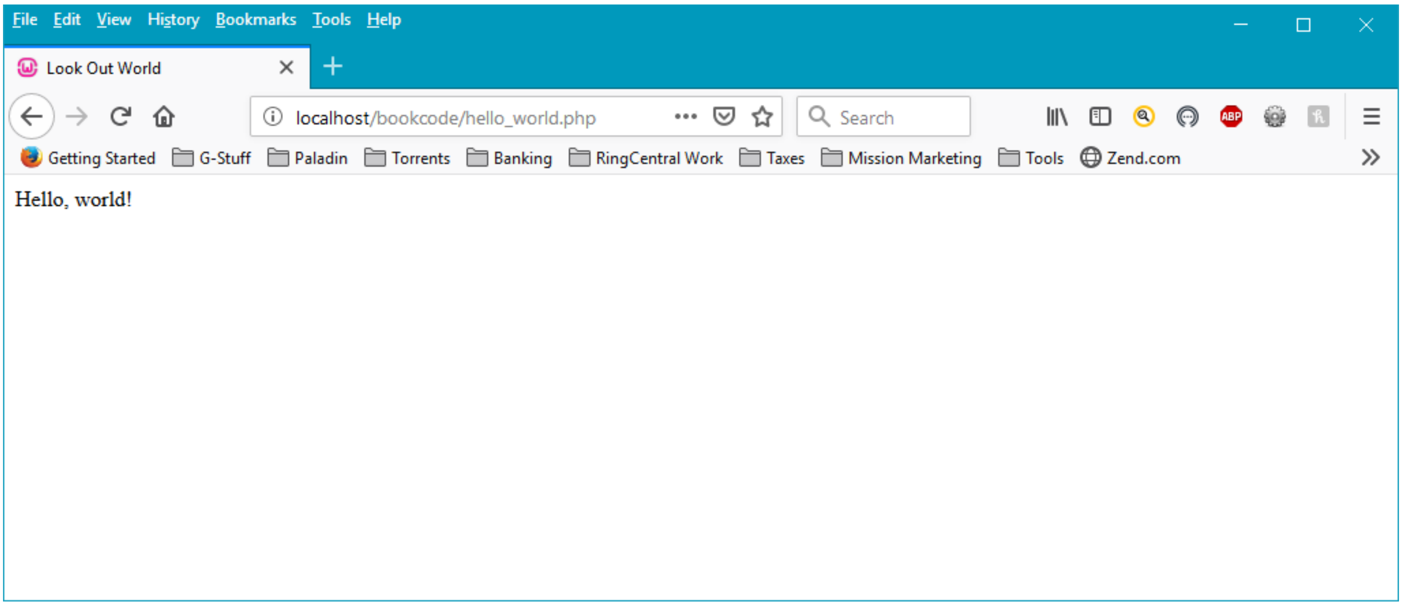
جولة في PHP

صفحات PHP هي بشكل عام صفحات HTML تحتوي على أوامر PHP مضمنة فيها. هذا على عكس العديد من حلول صفحات الويب الديناميكية الأخرى، وهي عبارة عن برامج نصية تنشئ HTML. يعالج خادم الويب أوامر PHP ويرسل مخرجاتها (وأي HTML من الملف) إلى المتصفح. يوضح المثال 1-1 صفحة PHP كاملة.

مثال 1-1. hello_world.php

```
<html>
 <head>
 <title>Look Out World</title>
 </head>
 <body>
 <?php echo "Hello, world!"; ?>
 </body>
</html>
```

احفظ محتويات المثال 1-1 في ملف، hello_world.php، وقم بتوجيه المتصفح إليه. تظهر النتائج في الشكل 1-2.



الشكل 1-2. إخراج hello_world.php

ينتج أمر PHP echo إخراج (السلسلة "Hello، world!" في هذه الحالة) يتم إدراجها في ملف HTML. في هذا المثال، يتم وضع كود PHP بين علامتي <?php و ?>. هناك طرق أخرى لوضع علامة على كود PHP الخاص بك - راجع الفصل 2 للحصول على وصف كامل.

صفحة التكوين

دالة ()phpinfo تنشئ صفحة HTML مليئة بالمعلومات حول كيفية تثبيت PHP وتهيئته حاليًا. يمكنك استخدامها لمعرفة ما إذا كان لديك امتدادات معينة مثبتة، أو ما إذا كان ملف php.ini قد تم تخصيصه. المثال 1-2 عبارة عن صفحة كاملة تعرض صفحة ()phpinfo.

المثال 1-2. باستخدام ()phpinfo

```
<?php phpinfo();?>
```

يوضح الشكل 1-3 الجزء الأول من ناتج المثال 1-2.

PHP Version 7.4.0

System	Windows NT TOWERCASE 10.0 build 18362 (Windows 10) AMD64
Build Date	Nov 27 2019 10:07:05
Compiler	Visual C++ 2017
Architecture	x64
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	D:\wamp64\bin\apache\apache2.4.41\bin\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902,TS,VC15
PHP Extension Build	API20190902,TS,VC15
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, ftps, phar
Registered Stream Socket Transports	tcp, udp, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.4.0, Copyright (c) Zend Technologies
with Zend OPcache v7.4.0, Copyright (c), by Zend Technologies
with Xdebug v2.8.0, Copyright (c) 2002-2019, by Derick Rethans

الشكل 1-3. الإخراج الجزئي لـ *phpinfo*()

نماذج forms

يقوم المثال 1-3 بإنشاء نموذج ومعالجته. عندما يرسل المستخدم النموذج، يتم إرسال المعلومات المكتوبة في حقل الاسم مرة أخرى إلى هذه الصفحة عبر إجراء النموذج $_SERVER['PHP_SELF']. يختبر كود PHP حقل اسم ويعرض ترحيباً إذا وجد واحداً.

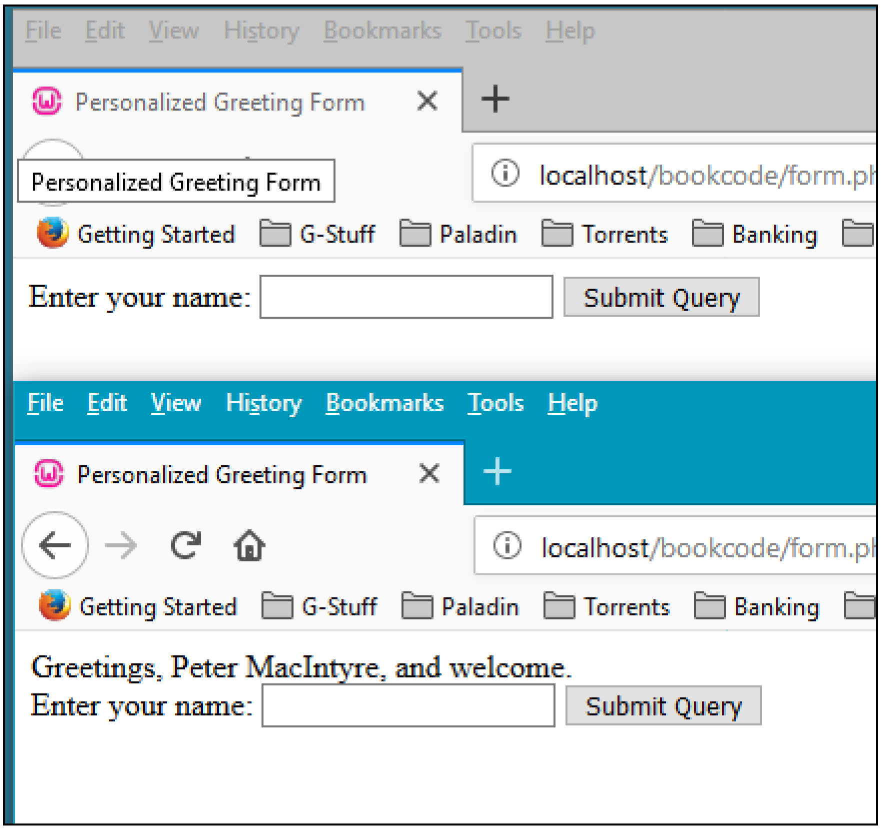
مثال 1-3. معالجة نموذج (form.php)

```
<html>
 <head>
 <title>Personalized Greeting Form</title>
 </head>

 <body>
 <?php if(!empty($_POST['name'])) {
 echo "Greetings, {$_POST['name']}, and welcome.";
 } ?>

 <form   action="<?php   echo   $_SERVER['PHP_SELF'];   ?>"
method="post">
 Enter your name: <input type="text" name="name" />
 <input type="submit" />
 </form>
 </body>
</html>
```

يظهر النموذج والرسالة في الشكل 1-4.



الشكل 1-4. صفحة النموذج والتحية

تصل برامج PHP إلى قيم النموذج بشكل أساسي من خلال متغيرات المصفوفة $_POST و $_GET. الفصل 8 يناقش النماذج ومعالجة النماذج بمزيد من التفصيل.

قواعد بيانات

يدعم PHP جميع أنظمة قواعد البيانات الشائعة، بما في ذلك قواعد البيانات المتوافقة مع MySQL و PostgreSQL و Oracle و Sybase و SQLite و ODBC. يوضح الشكل 1-5 جزءًا من استعلام قاعدة بيانات MySQL يتم تشغيله من خلال برنامج نصي PHP، ويعرض نتائج بحث عن كتاب على موقع لمراجعة الكتب. يسرد عنوان الكتاب وسنة نشر الكتاب ورقم ISBN للكتاب.

These Books are currently available		
Title	Year Published	ISBN
Executive Orders	1996	0-425-15863-2
Forward the Foundation	1993	0-553-56507-9
Foundation	1951	0-553-80371-9
Foundation and Empire	1952	0-553-29337-0
Foundation's Edge	1982	0-553-29338-9
I, Robot	1950	0-553-29438-5
Isaac Asimov: Gold	1995	0-06-055652-8
Rainbow Six	1998	0-425-17034-9
Roots	1974	0-440-17464-3
Second Foundation	1953	0-553-29336-2
Teeth of the Tiger	2003	0-399-15079-X
The Best of Isaac Asimov	1973	0-449-20829-X
The Hobbit	1937	0-261-10221-4
The Return of The King	1955	0-261-10237-0
The Sum of All Fears	1991	0-425-13354-0
The Two Towers	1954	0-261-10236-2

الشكل 1-5. يتم تشغيل استعلام قائمة كتب MySQL من خلال نص PHP

يتصل الكود الموجود في المثال 1-4 بقاعدة البيانات، ويصدر استعلامًا لاسترداد جميع الكتب المتاحة (مع عبارة WHERE)، وينتج جدولًا كمخرج لجميع النتائج التي تم إرجاعها من خلال حلقة while.

ملاحظة:

كود SQL لقاعدة البيانات هذه موجود في library.sql الملف المتوفر. يمكنك إسقاط هذا الكود في MySQL بعد إنشاء قاعدة بيانات المكتبة والحصول على نموذج قاعدة البيانات تحت تصرفك لاختبار نموذج التعليمات البرمجية التالي بالإضافة إلى العينات ذات الصلة في الفصل 9.

مثال 1-4. الاستعلام عن قاعدة بيانات الكتب (booklist.php)

```
<?php

$db = new mysqli("localhost", "petermac", "password",
"library");

// make sure the above credentials are correct for your
environment
if ($db->connect_error) {
 die("Connect Error ({$db->connect_errno}) {$db-
>connect_error}");
}

$sql = "SELECT * FROM books WHERE available = 1 ORDER BY title";
$result = $db->query($sql);
```

```
?>
<html>
<body>

<table cellSpacing="2" cellPadding="6" align="center"
border="1">
 <tr>
 <td colspan="4">
 <h3 align="center">These Books are currently available</h3>
 </td>
 </tr>

 <tr>
 <td align="center">Title</td>
 <td align="center">Year Published</td>
 <td align="center">ISBN</td>
 </tr>
 <?php while ($row = $result->fetch_assoc()) { ?>
 <tr>
 <td><?php echo stripslashes($row['title']); ?></td>
 <td align="center"><?php echo $row['pub_year']; ?></td>
 <td><?php echo $row['ISBN']; ?></td>
 </tr>
 <?php } ?>
</table>

</body>
</html>
```

يحرك المحتوى الديناميكي الذي توفره قاعدة البيانات الأخبار والمدونات ومواقع التجارة الإلكترونية في قلب الويب. مزيد من التفاصيل حول الوصول إلى قواعد البيانات من PHP موجودة في الفصل 9.

الرسومات

باستخدام PHP، يمكنك بسهولة إنشاء الصور ومعالجتها باستخدام امتداد GD. يوفر المثال 1-5 حقل إدخال نص يتيح للمستخدم تحديد نص الزر. يأخذ ملف صورة زر فارغًا، ويقوم بتوسيط النص الذي تم تمريره على أنه "رسالة" معلمة GET. ثم يتم إرسال النتيجة إلى المتصفح كصورة PNG.

مثال 1-5. الأزرار الديناميكية (Graphic_example.php)

```
<?php
if (isset($_GET['message'])) {
 // load font and image, calculate width of text
 $font = dirname(__FILE__) . '/fonts/blazed.ttf';
 $size = 12;
 $image = imagecreatefrompng("button.png");
 $tsize = imagettfbbox($size, 0, $font, $_GET['message']);

 // center
 $dx = abs($tsize[2] - $tsize[0]);
 $dy = abs($tsize[5] - $tsize[3]);
 $x = (imagesx($image) - $dx) / 2;
 $y = (imagesy($image) - $dy) / 2 + $dy;

 // draw text
 $black = imagecolorallocate($im,0,0,0);
```

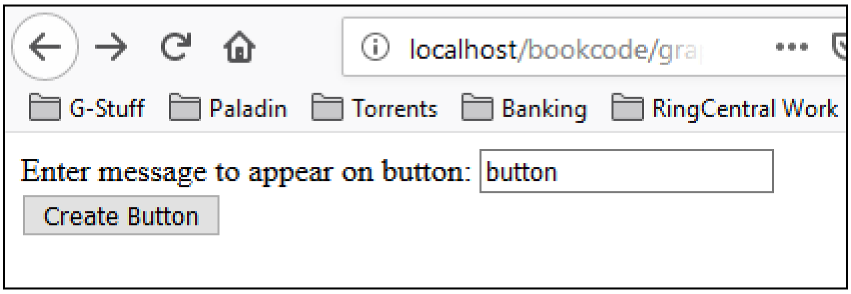
```
 imagettftext($image, $size, 0, $x, $y, $black, $font,
$_GET['message']);

 // return image
 header("Content-type: image/png");
 imagepng($image);

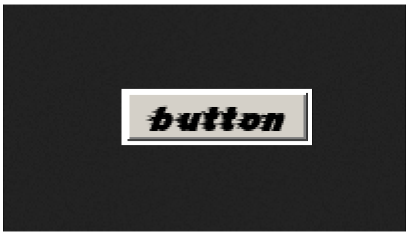
 exit;
} ?>
<html>
 <head>
 <title>Button Form</title>
 </head>

 <body>
 <form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
 Enter message to appear on button:
 <input type="text" name="message" /><br />
 <input type="submit" value="Create Button" />
 </form>
 </body>
</html>
```

يظهر النموذج الذي تم إنشاؤه بواسطة المثال 1-5 في الشكل 1-6. يظهر الزر الذي تم إنشاؤه في الشكل 1-7.



الشكل 1-6. شكل إنشاء زر



الشكل 1-7. تم إنشاء الزر

يمكنك استخدام GD لتغيير حجم الصور ديناميكيًا وإنتاج الرسوم البيانية وغير ذلك الكثير. تحتوي PHP أيضًا على عدة امتدادات لإنشاء مستندات بتنسيق PDF الشهير من Adobe. يغطي الفصل 10 إنشاء الصور الديناميكية بعمق، بينما يقدم الفصل 11 إرشادات حول كيفية إنشاء ملفات Adobe PDF.

ماذا بعد

الآن بعد أن تعرفت على ما هو ممكن باستخدام PHP، فأنت جاهز لتعلم كيفية البرمجة باللغة. نبدأ بهيكلها الأساسي، مع التركيز بشكل خاص على الوظائف التي يحددها المستخدم، ومعالجة السلسلة، والبرمجة الموجهة للكائنات. ثم ننتقل إلى مناطق تطبيق محددة، مثل الويب وقواعد البيانات والرسومات و XML والأمان. ننتهي بمراجع سريعة للوظائف المدمجة والإضافات. أتقن هذه الفصول، وستكون قد أتقنت لغة PHP!

الفصل الثاني أساسيات اللغة

يقدم هذا الفصل جولة سريعة في لغة PHP الأساسية، تغطي الموضوعات الأساسية مثل أنواع البيانات والمتغيرات والعمليات وبيانات التحكم في التدفق. تتأثر لغة PHP بشدة بلغات البرمجة الأخرى، مثل Perl و C، لذلك إذا كانت لديك خبرة في استخدام هذه اللغات، فيجب أن تكون لغة PHP سهلة التعلم. إذا كانت PHP من أولى لغات البرمجة لديك، فلا داعي للذعر. نبدأ بالوحدات الأساسية لبرنامج PHP ونبني معرفتك من هناك.

البنية المعجمية Lexical Structure

الهيكل المعجمي للغة البرمجة هو مجموعة القواعد الأساسية التي تحكم كيفية كتابة البرامج بتلك اللغة. إنه بناء الجملة الأقل مستوى للغة ويحدد أشياء مثل شكل أسماء المتغيرات، وما هي الأحرف المستخدمة للتعليقات، وكيف يتم فصل عبارات البرنامج عن بعضها البعض.

الحساسية لحالة الأحرف

أسماء الفئات والدوال المعرفة من قبل المستخدم، وكذلك التركيبات والكلمات الرئيسية المضمنة (مثل echo، while، class ، إلخ) ، غير حساسة لحالة الأحرف. وبالتالي، فإن هذه الأسطر الثلاثة متساوية:

```
echo("hello, world");
ECHO("hello, world");
EcHo("hello, world");
```

من ناحية أخرى، فإن المتغيرات حساسة لحالة الأحرف. وهذا يعني أن $name و $NaME و $NAME هي ثلاثة متغيرات مختلفة.

الجمل والفواصل المنقوطة

الجملة عبارة عن مجموعة من كود PHP يقوم بشيء ما. يمكن أن تكون بسيطة مثل تخصيص متغير أو معقدة complicated مثل: حلقة مع نقاط خروج متعددة. فيما يلي عينة صغيرة من جمل PHP، بما في ذلك استدعاءات الدوال، وبعض تخصيصات البيانات المتغيرة، وجمل if:

```
echo "Hello, world";
myFunction(42, "O'Reilly");
$a = 1;
$name = "Elphaba";
$b = $a / 25.0;
if ($a == $b) {
 echo "Rhyme? And Reason?";
}
```

تستخدم PHP الفاصلة المنقوطة للفصل بين الجمل البسيطة. لا تحتاج الجمل المركبة التي تستخدم الأقواس المتعرجة لتمييز كتلة من التعليمات البرامجية، مثل اختبار أو حلقة شرطية، إلى فاصلة منقوطة بعد قوس إغلاق. بخلاف اللغات الأخرى، في PHP، لا تعد الفاصلة المنقوطة قبل قوس الإغلاق اختيارية:

```
if ($needed) {
 echo "We must have it!"; // semicolon required here
} // no semicolon required here after the brace
```

ومع ذلك، فإن الفاصلة المنقوطة اختيارية قبل علامة إغلاق PHP:

```
<?php
```

```php
if ($a == $b) {
 echo "Rhyme? And Reason?";
}
echo "Hello, world" // no semicolon required before closing tag
?>
```

من الممارسات البرمجية الجيدة تضمين الفواصل المنقوطة الاختيارية، لأنها تسهل إضافة الكود لاحقًا.

المسافة البيضاء وفواصل الأسطر

بشكل عام، لا تهم المسافات البيضاء في برنامج PHP. يمكنك نشر جمل عبر أي عدد من الأسطر، أو تجميع مجموعة من الجمل معًا في سطر واحد. على سبيل المثال، هذه الجملة:

```php
raisePrices($inventory, $inflation, $costOfLiving, $greed);
```

يمكن أيضًا كتابتها بمزيد من المسافات:

```php
raisePrices (
 $inventory ,
 $inflation ,
 $costOfLiving ,
 $greed
) ;
```

أو بمسافة بيضاء أقل:

```
raisePrices($inventory,$inflation,$costOfLiving,$greed);
```

يمكنك الاستفادة من هذا التنسيق المرن لجعل شفرتك أكثر قابلية للقراءة (من خلال ترتيب المهام، والمسافة البادئة، وما إلى ذلك). يستفيد بعض المبرمجين البطيئين من هذا التنسيق الحر وينشئون كوداً غير قابل للقراءة تماماً - وهذا غير مستحسن.

تعليقات

تعطي التعليقات معلومات للأشخاص الذين قرأوا التعليمات البرمجية الخاصة بك، ولكن يتم تجاهلها بواسطة PHP في وقت التنفيذ. حتى إذا كنت تعتقد أنك الشخص الوحيد الذي سيقرأ شفرتك على الإطلاق، فمن الجيد تضمين التعليقات في شفرتك - في وقت لاحق، يمكن أن يبدو الرمز الذي كتبته منذ شهور بسهولة كما لو أنه كتبه شخص غريب.

من الممارسات الجيدة أن تجعل تعليقاتك قليلة بما يكفي حتى لا تعترض طريق الشفرة نفسها ولكن وفيرة بما يكفي بحيث يمكنك استخدام التعليقات لمعرفة ما يحدث. لا تعلق على أشياء واضحة، لئلا تدفن التعليقات التي تصف أشياء صعبة. على سبيل المثال، هذا لا قيمة له:

```
$x = 17; // store 17 into the variable $x
```

بينما التعليقات على هذا التعبير العادي المعقد ستساعد كل من يحافظ على الكود الخاص بك:

```
// convert &#nnn; entities into characters

$text = preg_replace('/&#([0-9])+;/', "chr('\\1')", $text);
```

توفر PHP عدة طرق لتضمين التعليقات في التعليمات البرمجية الخاصة بك، وكلها مستعارة من اللغات الحالية مثل C و C++ و Unix shell. بشكل عام، استخدم التعليقات ذات النمط C للتعليق على الكود، والتعليقات ذات النمط C ++ للتعليق على الكود.

تعليقات SHELL-STYLE

عندما تصادف PHP رمز علامة التجزئة (#) داخل الكود، فإن كل شيء بدءًا من علامة التجزئة إلى نهاية السطر أو نهاية قسم كود PHP (أيهما يأتي أولاً) يعتبر تعليقًا. توجد طريقة التعليق هذه في لغات البرمجة النصية لـ Unix shell وهي مفيدة في وضع تعليقات توضيحية على أسطر مفردة من التعليمات البرمجية أو عمل ملاحظات قصيرة.

نظراً لأن علامة التجزئة مرئية على الصفحة، يتم استخدام تعليقات نمط الصدفة أحياناً لتمييز كل التعليمات البرمجية:

```
#######################
## Cookie functions
#######################
```

يتم استخدامها في بعض الأحيان قبل سطر من التعليمات البرمجية لتحديد ما يفعله هذا الرمز، وفي هذه الحالة يتم عادةً وضع مسافة بادئة لها على نفس مستوى الشفرة التي تم إعداد التعليق لها:

```
if ($doubleCheck) {
 # create an HTML form requesting that the user confirm the action
 echo confirmationForm();
}
```

غالبًا ما يتم وضع التعليقات القصيرة على سطر واحد من التعليمات البرمجية في نفس سطر الكود:

```
$value = $p * exp($r * $t); # calculate compounded interest
```

عندما تقوم بخلط كود HTML و PHP بإحكام، قد يكون من المفيد أن تقوم علامة PHP المغلقة بإنهاء التعليق:

```
<?php $d = 4; # Set $d to 4. ?> Then another <?php echo $d; ?>
Then another 4
```

تعليقات ++C

عندما تصادف PHP شريحتين مائلتين (//) داخل الكود، فإن كل شيء بدءًا من الشرطات المائلة إلى نهاية السطر أو نهاية قسم الكود، أيهما يأتي أولاً، يعتبر تعليقًا. طريقة التعليق هذه مشتقة من ++C. والنتيجة هي نفس نمط تعليق الصدفة.

فيما يلي أمثلة للتعليقات على نمط الصدفة، أعيد كتابتها لاستخدام تعليقات ++C:

```
////////////////////////
// Cookie functions
////////////////////////

if ($doubleCheck) {
 // create an HTML form requesting that the user confirm the action
 echo confirmationForm();
}

$value = $p * exp($r * $t); // calculate compounded interest
```

```
<?php $d = 4; // Set $d to 4. ?> Then another <?php echo $d; ?>
Then another 4
```

تعليقات C

في حين أن التعليقات بنمط shell و C++ مفيدة للتعليق التوضيحي للكود أو لعمل ملاحظات قصيرة، تتطلب التعليقات الأطول أسلوبًا مختلفًا. لذلك، تدعم PHP تعليقات الكتلة التي يأتي تركيبها من لغة البرمجة C. عندما تصادف PHP شرطة مائلة متبوعة بعلامة النجمة (/*)، فإن كل شيء بعد ذلك، حتى يصادف علامة النجمة متبوعة بشرطة مائلة (*/) ، يعتبر تعليقًا. هذا النوع من التعليقات، على عكس تلك الموضحة سابقًا، يمكن أن يمتد إلى سطور متعددة.

في ما يلي مثال على تعليق متعدد الأسطر على نمط C:

```
/* In this section, we take a bunch of variables and
 assign numbers to them. There is no real reason to
 do this, we're just having fun.
*/
$a = 1;
$b = 2;
$c = 3;
$d = 4;
```

نظرًا لأن التعليقات على النمط C لها علامات بداية ونهاية محددة، يمكنك دمجها بإحكام مع التعليمات البرمجية. يميل هذا إلى جعل قراءة التعليمات البرمجية أكثر صعوبة ولا يُنصح به:

```
/* These comments can be mixed with code too,
see? */ $e = 5; /* This works just fine. */
```

يمكن أن تستمر تعليقات نمط C، على عكس الأنواع الأخرى، بعد علامات علامات PHP النهائية. فمثلا:

```
<?php
$l = 12;
$m = 13;
/* A comment begins here
?>
<p>Some stuff you want to be HTML.</p>
<?= $n = 14; ?>
*/
echo("l=$l m=$m n=$n\n");
?><p>Now <b>this</b> is regular HTML...</p>
l=12 m=13 n=
<p>Now <b>this</b> is regular HTML...</p>
```

يمكنك وضع مسافة بادئة للتعليقات كما تريد:

```
/* There are no
 special indenting or spacing
 rules that have to be followed, either.



 */
```

يمكن أن تكون التعليقات على النمط C مفيدة لتعطيل أقسام التعليمات البرمجية. في المثال التالي، قمنا بتعطيل الجملتين الثانية والثالثة، بالإضافة إلى التعليق المضمن، من خلال تضمينهما في تعليق الحظر. لتمكين الكود، كل ما يتعين علينا القيام به هو إزالة علامات التعليق:

```
$f = 6;
/*
```

```
$g = 7; # This is a different style of comment
$h = 8;
*/
```

ومع ذلك، يجب أن تكون حريصًا على عدم محاولة دمج التعليقات:

```
$i = 9;
/*
$j = 10; /* This is a comment */
$k = 11;
Here is some comment text.
*/
```

في هذه الحالة، تحاول PHP (وتفشل) تنفيذ الجملة (non). إليك بعض نص التعليق وإرجاع خطأ.

حرفية

الحرفي هو قيمة البيانات التي تظهر مباشرة في البرنامج. فيما يلي جميع القيم الحرفية في PHP:

```
2001
0xFE
1.4142
"Hello World"
'Hi'
true
null
```

معرفات

المعرف هو مجرد اسم. في PHP، تُستخدم المعرفات لتسمية المتغيرات والدوال والثوابت والفئات. يجب أن يكون الحرف الأول للمعرّف حرف ASCII (أحرف كبيرة أو صغيرة)، أو حرف الشرطة السفلية (_)، أو أي من الأحرف بين ASCII 0x7F و ASCII 0xFF. بعد الحرف الأولي، هذه الأحرف والأرقام من 0 إلى 9 صالحة.

أسماء المتغيرات

تبدأ أسماء المتغيرات دائمًا بعلامة الدولار ($) وتكون حساسة لحالة الأحرف. فيما يلي بعض أسماء المتغيرات الصالحة:

```
$bill
$head_count
$MaximumForce
$I_HEART_PHP
$_underscore
$_int
```

فيما يلي بعض أسماء المتغيرات غير القانونية:

```
$not valid
$|
$3wa
```

هذه المتغيرات كلها مختلفة بسبب حساسية حالة الأحرف:

```
$hot_stuff $Hot_stuff $hot_Stuff $HOT_STUFF
```

أسماء الدوال

أسماء الدوال ليست حساسة لحالة الأحرف (تمت مناقشة الدوال بمزيد من التفصيل في الفصل 3). فيما يلي بعض أسماء الدوال الصالحة:

```
tally
list_all_users
deleteTclFiles
LOWERCASE_IS_FOR_WIMPS
_hide
```

تشير جميع أسماء الدوال هذه إلى نفس الدالة:

```
howdy HoWdY HOWDY HOWdy howdy
```

أسماء الفصل CLASS

تتبع أسماء الفئات القواعد القياسية لمعرّفات PHP كما أنها ليست حساسة لحالة الأحرف. فيما يلي بعض أسماء الفئات الصالحة:

```
Person
account
```

اسم الفئة `stdClass` هو اسم فئة محجوز.

الثوابت

الثابت هو معرف لقيمة لن تتغير، يمكن أن تكون القيم العددية (منطقية، وعدد صحيح، ومزدوجة، وسلسلة) والمصفوفات ثوابت. بمجرد التعيين، لا يمكن أن تتغير قيمة الثابت. تتم الإشارة إلى الثوابت بواسطة معرفاتها ويتم تعيينها باستخدام دالة `define()`:

```
define('PUBLISHER', "O'Reilly Media");
echo PUBLISHER;
```

الكلمات الرئيسية

الكلمة الرئيسية keyword (أو الكلمة المحجوزة reserved word) هي كلمة تخصصها اللغة لدوالها الأساسية - لا يمكنك إعطاء دالة أو فئة أو ثابت نفس اسم كلمة رئيسية. يسرد الجدول 2-1 الكلمات الرئيسية في PHP، والتي لا تتأثر بحالة الأحرف.

الجدول 2-1. الكلمات الرئيسية للغة PHP الأساسية

```
__CLASS__
__DIR__
__FILE__
__FUNCTION__
__LINE__
__METHOD__
__NAMESPACE__
__TRAIT__
__halt_compiler()
abstract
and
array()
as
break
callable
case
catch
class
clone
const
continue
declare
default
die()
do
echo
else
elseif
empty()
enddeclare
endfor
endforeach
endif
endswitch
endwhile
eval()
exit()
extends
final
finally
for
foreach
function
global
goto
if
implements
include
include_once
instanceof
insteadof
interface
isset()
list()
namespace
new
or
print
private
protected
public
require
require_once
return
static
switch
throw
trait
try
unset()
use
var
while
xor
yield
yield from
```

بالإضافة إلى ذلك، لا يمكنك استخدام معرّف مماثل لدالة PHP المدمجة. للحصول على قائمة كاملة بها.

أنواع البيانات

توفر PHP ثمانية أنواع من القيم أو أنواع البيانات. أربعة أنواع عددية (ذات قيمة واحدة): الأعداد الصحيحة، وأرقام الفاصلة العائمة، والسلاسل، والمنطقية. هناك نوعان من أنواع (التجميع) المركبة: المصفوفات والكائنات. النوعان المتبقيان هما نوعان خاصان: resource و NULL. تتم مناقشة الأرقام والمنطقية والموارد و NULL بالكامل هنا، بينما تعد السلاسل والمصفوفات والكائنات موضوعات كبيرة بما يكفي للحصول على الفصول الخاصة بها (الفصول 4 و 5 و 6 على التوالي).

عدد صحيح

الأعداد الصحيحة هي أرقام صحيحة، مثل 1 و 12 و 256. يختلف نطاق القيم المقبولة وفقًا لتفاصيل النظام الأساسي الخاص بك ولكنه يمتد عادةً من -2،147،483،648 إلى +2،147،483،647. على وجه التحديد، فإن النطاق يكافئ نطاق نوع البيانات الطويلة لمترجم سي الخاص بك. لسوء الحظ، لا يحدد معيار C النطاق الذي يجب أن يشتمل عليه هذا النوع الطويل، لذلك قد ترى في بعض الأنظمة نطاقًا صحيحًا مختلفًا.

يمكن كتابة القيم الحرفية الصحيحة بالنظام العشري أو الثماني أو الثنائي أو السادس عشري. يتم تمثيل القيم العشرية بتسلسل من الأرقام، بدون أصفار بادئة. قد يبدأ التسلسل بعلامة زائد (+) أو ناقص (-). إذا لم تكن هناك علامة، يفترض وجود إيجابي. تتضمن أمثلة الأعداد الصحيحة العشرية ما يلي:

```
1998
-641
+33
```

تتكون الأرقام الثمانية من صفر بادئ وسلسلة من الأرقام من 0 إلى 7. مثل الأرقام العشرية، يمكن أن تبدأ الأرقام الثمانية بعلامة زائد أو ناقص. فيما يلي بعض الأمثلة على القيم الثمانية والقيم العشرية المكافئة لها:

```
0755 // decimal 493
+010 // decimal 8
```

تبدأ القيم السداسية العشرية بـ 0x، متبوعة بسلسلة من الأرقام (0-9) أو بالحروف (A – F). يمكن أن تكون الأحرف كبيرة أو صغيرة ولكنها عادة ما تكون مكتوبة بأحرف كبيرة. كما هو الحال مع القيم العشرية والثمانية، يمكنك تضمين علامة بأرقام سداسية عشرية:

```
0xFF // decimal 255
0x10 // decimal 16
-0xDAD1 // decimal -56017
```

تبدأ الأعداد الثنائية بالرقم 0 ب، متبوعًا بتسلسل من الأرقام (0 و 1). كما هو الحال مع القيم الأخرى، يمكنك تضمين علامة في الأرقام الثنائية:

```
0b01100000 // decimal 96
0b00000010 // decimal 2
-0b10 // decimal -2
```

إذا حاولت تخزين متغير كبير جدًا بحيث لا يمكن تخزينه كعدد صحيح أو ليس عددًا صحيحًا، فسيتم تحويله تلقائيًا إلى رقم فاصلة عائمة.

استخدم الدالة is_int() (أو الاسم المستعار is_integer() الخاص بها) لاختبار ما إذا كانت القيمة عددًا صحيحًا:

```
if (is_int($x)) {
 // $x is an integer
}
```

أرقام النقطة العائمة

تمثل أرقام الفاصلة العائمة (يشار إليها غالبًا بالأرقام "الحقيقية") قيمًا رقمية بأرقام عشرية. مثل الأعداد الصحيحة، تعتمد حدودها على تفاصيل جهازك. أرقام الفاصلة العائمة في PHP تكافئ نطاق نوع البيانات المزدوج لمترجم سي الخاص بك. عادة، يسمح هذا بالأرقام بين 1.7E−308 و 1.7E+308 بدقة 15 رقمًا. إذا كنت بحاجة إلى مزيد من الدقة أو نطاق أوسع من قيم الأعداد الصحيحة، فيمكنك استخدام امتدادات BC أو GMP.

تتعرف PHP على أرقام الفاصلة العائمة المكتوبة بتنسيقين مختلفين. هناك النوع الذي نستخدمه جميعًا كل يوم:

```
3.14
0.017
-7.1
```

لكن PHP تتعرف أيضًا على الأرقام في الترميز العلمي:

```
0.314E1 // 0.314*10^1, or 3.14
17.0E-3 // 17.0*10^(-3), or 0.017
```

قيم الفاصلة العائمة هي فقط تمثيلات تقريبية للأرقام. على سبيل المثال، في العديد من الأنظمة، يتم تمثيل 3.5 بالفعل كـ 3.4999999999. هذا يعني أنه يجب عليك الحرص على تجنب كتابة التعليمات البرمجية التي تفترض تمثيل أرقام الفاصلة العائمة بدقة كاملة، مثل المقارنة المباشرة بين قيمتين للفاصلة العائمة باستخدام ==. النهج العادي هو المقارنة بعدة خانات عشرية:

```
if (intval($a * 1000) == intval($b * 1000)) {
  // numbers equal to three decimal places
}
```

استخدم الدالة is_float() (أو الاسم المستعار is_real() الخاص بها) لاختبار ما إذا كانت القيمة رقم فاصلة عائمة:

```
if (is_float($x)) {
  // $x is a floating-point number
}
```

سلاسل

نظرًا لأن السلاسل شائعة جدًا في تطبيقات الويب، تتضمن PHP دعمًا على المستوى الأساسي لإنشاء السلاسل ومعالجتها. السلسلة عبارة عن سلسلة من الأحرف ذات طول عشوائي. يتم تحديد القيم الحرفية للسلسلة بعلامات اقتباس مفردة أو مزدوجة:

```
'big dog'
"fat hog"
```

يتم توسيع المتغيرات (interpolated) داخل علامات الاقتباس المزدوجة، بينما لا يتم توسيع المتغيرات داخل علامات الاقتباس المفردة:

```
$name = "Guido";
echo "Hi, $name <br/>";
```

```
echo 'Hi, $name';
Hi, Guido
Hi, $name
```

تدعم علامات الاقتباس المزدوجة أيضًا مجموعة متنوعة من عمليات الهروب من السلسلة، كما هو موضح في الجدول 2-2.

الجدول 2-2. الهروب من التسلسلات في سلاسل ذات علامات اقتباس مزدوجة

Escape sequence	**Character represented**
`\"`	Double quotes
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\\`	Backslash
`\$`	Dollar sign
`\{`	Left brace
`\}`	Right brace
`\[`	Left bracket
`\]`	Right bracket
`\0` through `\777`	ASCII character represented by octal value
`\x0` through `\xFF`	ASCII character represented by hex value

تتعرف سلسلة ذات علامات اقتباس مفردة على \\ للحصول على شرطة مائلة عكسية و \' للحصول على اقتباس مفرد حرفي:

```
$dosPath = 'C:\\WINDOWS\\SYSTEM';

$publisher = 'Tim O\'Reilly';

echo "$dosPath $publisher";

C:\WINDOWS\SYSTEM Tim O'Reilly
```

لاختبار ما إذا كانت سلسلتان متساويتين، استخدم عامل المقارنة == (مزدوج يساوي):

```
if ($a == $b) {
 echo "a and b are equal";
}
```

استخدم الدالة `is_string()` لاختبار ما إذا كانت القيمة سلسلة أم لا:

```
if (is_string($x)) {
 // $x is a string
}
```

توفر PHP عمليات حسابية ودوال لمقارنة السلاسل وتفكيكها وتجميعها والبحث عنها واستبدالها وتقليمها، بالإضافة إلى مجموعة من دوال السلاسل المتخصصة للعمل مع ترميز HTTP و HTML و SQL. نظراً لوجود العديد من دوال معالجة السلاسل، فقد خصصنا فصلًا كاملاً (الفصل 4) لتغطية جميع التفاصيل.

قيمة منطقية

تمثل القيمة المنطقية قيمة *truth* - فهي توضح ما إذا كان الشيء صحيحًا أم لا. مثل معظم لغات البرمجة، تعرّف PHP بعض القيم على أنها true والبعض الآخر false. الصدق والخطأ يحددان نتيجة الكود الشرطي مثل:

```
if ($alive) { ... }
```

في PHP، يتم تقييم جميع القيم التالية على أنها false:

- ❖ الكلمة الرئيسية false
- ❖ العدد الصحيح 0
- ❖ قيمة النقطة العائمة 0.0
- ❖ السلسلة الفارغة ("") والسلسلة "0"
- ❖ مصفوفة بعناصر صفرية
- ❖ القيمة NULL

القيمة غير الصحيحة هي القيمة الصحيحة، بما في ذلك جميع قيم الموارد (التي سيتم وصفها لاحقًا في قسم "الموارد").

توفر PHP كلمات رئيسية true وfalse للتوضيح:

```
$x = 5; // $x has a true value
$x = true; // clearer way to write it
$y = ""; // $y has a false value
$y = false; // clearer way to write it
```

استخدم الدالة is_bool() لاختبار ما إذا كانت القيمة منطقية:

```
if (is_bool($x)) {
 // $x is a Boolean
}
```

المصفوفات

تحتوي المصفوفة على مجموعة من القيم، والتي يمكنك تحديدها من خلال الموضع (رقم)، مع كون الصفر هو الموضع الأول أو بعض الأسماء التعريفية (سلسلة)، تسمى الفهرس الترابطي *"associative index"*:

```
$person[0] = "Edison";
$person[1] = "Wankel";
$person[2] = "Crapper";

$creator['Light bulb'] = "Edison";
$creator['Rotary Engine'] = "Wankel";
$creator['Toilet'] = "Crapper";
```

ينشئ بناء array() مصفوفة. إليك مثالين:

```
$person = array("Edison", "Wankel", "Crapper");
$creator = array('Light bulb' => "Edison",
 'Rotary Engine' => "Wankel",
 'Toilet' => "Crapper");
```

توجد عدة طرق للتكرار خلال المصفوفات، ولكن أكثرها شيوعًا هي حلقة foreach:

```
foreach ($person as $name) {
 echo "Hello, {$name}<br/>";
}

foreach ($creator as $invention => $inventor) {
 echo "{$inventor} invented the {$invention}<br/>";
}
```

```
Hello, Edison
Hello, Wankel
Hello, Crapper
Edison created the Light bulb
Wankel created the Rotary Engine
Crapper created the Toilet
```

يمكنك فرز عناصر المصفوفة بدوال الفرز المختلفة:

```
sort($person);
// $person is now array("Crapper", "Edison", "Wankel")

 asort($creator);
// $creator is now array('Toilet' => "Crapper",
// 'Light bulb' => "Edison",
// 'Rotary Engine' => "Wankel");
```

Use the is_array() function to test whether a value is an array:

```
if (is_array($x)) {
 // $x is an array
}
```

هناك دوال لإرجاع عدد العناصر في المصفوفة، وجلب كل قيمة في المصفوفة، وأكثر من ذلك بكثير. المصفوفات مغطاة بعمق في الفصل 5.

Objects

يدعم PHP أيضًا البرمجة الشيئية (OOP). يعزز OOP التصميم المعياري النظيف؛ يبسط التصحيح والصيانة؛ ويساعد في إعادة استخدام الكود. الفئات هي اللبنات الأساسية للتصميم الموجه للكائنات. الفئة عبارة عن تعريف للبنية التي تحتوي على خصائص (متغيرات) وطرق (دوال). يتم تحديد الفئات باستخدام الكلمة الأساسية class:

```
class Person
{
 public $name = '';

 function name ($newname = NULL)
 {
 if (!is_null($newname)) {
 $this->name = $newname;
 }

 return $this->name;
 }
}
```

بمجرد تحديد فئة، يمكن إنشاء أي عدد من الكائنات منها باستخدام الكلمة الرئيسية new، ويمكن الوصول إلى خصائص الكائن وطرقه باستخدام ->:

```
$ed = new Person;
$ed->name('Edison');
```

```
echo "Hello, {$ed->name} <br/>";
$tc = new Person;
$tc->name('Crapper');
echo "Look out below {$tc->name} <br/>";
```

```
Hello, Edison
Look out below Crapper
```

استخدم الدالة `is_object()` لاختبار ما إذا كانت القيمة كائنًا:

```
if (is_object($x)) {
 // $x is an object
}
```

يصف الفصل 6 الفئات والكائنات بمزيد من التفصيل، بما في ذلك الوراثة inheritance والتغليف encapsulation والاستبطان introspection.

مصادر أو موارد

توفر العديد من الوحدات العديد من الدوال للتعامل مع العالم الخارجي. على سبيل المثال، يحتوي كل ملحق قاعدة بيانات على الأقل على دالة للاتصال بقاعدة البيانات، ووظيفة للاستعلام عن قاعدة البيانات، ووظيفة لإغلاق الاتصال بقاعدة البيانات. نظراً لإمكانية فتح اتصالات قاعدة بيانات متعددة في وقت واحد، تمنحك دالة الاتصال شيئاً يمكن من خلاله تحديد هذا الاتصال الفريد عند استدعاء الاستعلام وإغلاق الدوال: مورد resource (أو مقبض *handle*).

لكل مورد نشط معرف فريد. كل معرف هو فهرس رقمي في جدول بحث PHP داخلي يحتوي على معلومات حول جميع الموارد النشطة. تحتفظ PHP بمعلومات حول كل مورد في هذا الجدول، بما في ذلك عدد المراجع (أو استخدامات) المورد في جميع أنحاء الكود. عندما يختفي المرجع الأخير لقيمة مورد، يتم استدعاء الملحق الذي أنشأ المورد لأداء مهام مثل تحرير أي ذاكرة أو إغلاق أي اتصال لهذا المورد:

```
$res = database_connect(); // fictitious database connect function
database_query($res);

$res = "boo";
// database connection automatically closed because $res is redefined
```

تظهر فائدة هذا التنظيف التلقائي بشكل أفضل في الدوال، عندما يتم تخصيص المورد لمتغير محلي. عندما تنتهي الدالة، تستعيد PHP قيمة المتغير:

```
function search() {
 $res = database_connect();
 database_query($res);
}
```

عند عدم وجود المزيد من الإشارات إلى المورد، يتم إيقافه تلقائيًا.

ومع ذلك، توفر معظم الإضافات إيقاف تشغيل محدد أو دالة إغلاق، ويُعتبر أسلوبًا جيدًا لاستدعاء هذه الدالة بشكل صريح عند الحاجة بدلاً من الاعتماد على النطاق المتغير لبدء عملية تنظيف الموارد.

استخدم الدالة `is_resource()` لاختبار ما إذا كانت القيمة موردًا:

```
if (is_resource($x)) {
 // $x is a resource
}
```

عمليات الاسترجاعات Callbacks

عمليات الاسترجاعات هي دوال أو طرق كائن مستخدمة بواسطة بعض الدوال، مثل: `call_user_func()`. يمكن أيضًا إنشاء عمليات رد النداء بطريقة `create_function()` ومن خلال عمليات الإغلاق (الموضحة في الفصل 3):

```
$callback = function()
{
 echo "callback achieved";
};

call_user_func($callback);
```

NULL

هناك قيمة واحدة فقط لنوع البيانات NULL. تتوفر هذه القيمة من خلال الكلمة الأساسية NULL غير حساسة لحالة الأحرف. تمثل القيمة NULL متغيرًا ليس له قيمة (مشابه لـ Perl's `undef` أو Python's `None`):

```
$aleph = "beta";
$aleph = null; // variable's value is gone
$aleph = Null; // same
```

```
$aleph = NULL; // same
```

استخدم الدالة `is_null()` لاختبار ما إذا كانت القيمة فارغة - على سبيل المثال، لمعرفة ما إذا كان للمتغير قيمة أم لا:

```
if (is_null($x)) {
 // $x is NULL
}
```

المتغيرات

المتغيرات في PHP هي معرفات مسبوقة بعلامة الدولار ($). فمثلا:

```
$name
$Age
$_debugging
$MAXIMUM_IMPACT
```

قد يحمل المتغير قيمة من أي نوع. لا يوجد وقت ترجمة أو نوع وقت تشغيل التحقق من المتغيرات. يمكنك استبدال قيمة متغير بأخرى من نوع مختلف:

```
$what = "Fred";
$what = 35;
$what = array("Fred", 35, "Wilma");
```

لا توجد صيغة واضحة للتصريح عن المتغيرات في PHP. في المرة الأولى التي يتم فيها تعيين قيمة المتغير، يتم إنشاء المتغير في الذاكرة. بمعنى آخر، تعيين قيمة لمتغير يعمل أيضًا كإعلان. على سبيل المثال، هذا برنامج PHP كامل صالح:

```
$day = 60 * 60 * 24;
echo "There are {$day} seconds in a day.";
There are 86400 seconds in a day.
```

المتغير الذي لم يتم تعيين قيمته يتصرف مثل القيمة NULL:

```
if ($uninitializedVariable === NULL) {
 echo "Yes!";
}
Yes!
```

المتغيرات المتغيرة Variable Variables

يمكنك الإشارة إلى قيمة المتغير الذي تم تخزين اسمه في متغير آخر عن طريق تقديم مرجع المتغير بعلامة دولار إضافية ($). فمثلا:

```
$foo = "bar";
$$foo = "baz";
```

بعد تنفيذ الجملة الثانية، المتغير $bar له القيمة "baz".

مراجع متغيرة Variable References

في PHP، المراجع هي كيفية إنشاء أسماء مستعارة أو مؤشرات متغيرة. لجعل `$black` اسمًا مستعارًا للمتغير `$white`، استخدم:

```
$black =& $white;
```

فُقدت القيمة القديمة لـ $black، إن وجدت. بدلاً من ذلك، أصبح $black الآن اسمًا آخر للقيمة المخزنة في $black:

```
$bigLongVariableName = "PHP";
$short =& $bigLongVariableName;
$bigLongVariableName .= " rocks!";
print "\$short is $short <br/>";
print "Long is $bigLongVariableName";
$short is PHP rocks!
Long is PHP rocks!
```

```
$short = "Programming $short";
print "\$short is $short <br/>";
print "Long is $bigLongVariableName";
$short is Programming PHP rocks!
Long is Programming PHP rocks!
```

بعد التخصيص، يكون المتغيرين اسمين بديلين لنفس القيمة. لا يؤثر عدم ضبط متغير يحمل اسمًا مستعارًا على الأسماء الأخرى لقيمة هذا المتغير، ومع ذلك:

```
$white = "snow";
$black =& $white;
unset($white);
print $black;
snow
```

يمكن أن تُرجع الدوال قيمًا بالمرجع (على سبيل المثال، لتجنب نسخ سلاسل أو مصفوفات كبيرة، كما تمت مناقشته في الفصل 3):

```
function &retRef() // note the &
{
 $var = "PHP";

 return $var;
}

$v =& retRef(); // note the &
```

نطاق المتغير

نطاق المتغير، الذي يتحكم فيه موقع إعلان المتغير، يحدد تلك الأجزاء من البرنامج التي يمكنها الوصول إليه. **هناك أربعة أنواع من نطاقات المتغير في PHP:** المحلي (local) والعالمي (global) والثابت (static) ومعلمات الدوال (function).

النطاق المحلي

المتغير المعلن في دالة محلي لتلك الدالة. أي أنه مرئي فقط للكود في تلك الدالة (باستثناء تعريفات الدوال المتداخلة "function definitions")؛ لا يمكن الوصول إليه خارج الدالة. بالإضافة إلى ذلك، بشكل افتراضي، لا يمكن الوصول إلى المتغيرات المحددة خارج دالة (تسمى المتغيرات العامة) داخل الدالة. على سبيل المثال، إليك دالة تعمل على تحديث متغير محلي بدلاً من متغير عام:

```
function updateCounter()
```

```
{
 $counter++;
}


$counter = 10;
updateCounter();


echo $counter;
10
```

`$counter` داخل الدالة محلي لهذه الدالة لأننا لم نقل غير ذلك. تعمل الدالة على زيادة متغير `$counter` الخاص بها، والذي يتم إتلافه عند انتهاء الإجراء الفرعي. يظل `$counter` العالمي مضبوطًا على 10.

فقط الدوال يمكن أن توفر النطاق المحلي. على عكس اللغات الأخرى، في PHP لا يمكنك إنشاء متغير يكون نطاقه عبارة عن حلقة أو فرع شرطي أو أي نوع آخر من الكتل.

النطاق العالمي

المتغيرات المعلنة خارج دالة تكون عالمية. أي أنه يمكن الوصول إليها من أي جزء من البرنامج. ومع ذلك، فهي غير متوفرة بشكل افتراضي داخل الدالة. للسماح لدالة ما بالوصول إلى متغير عام، يمكنك استخدام الكلمة الأساسية `global` داخل الدالة لتعريف المتغير داخل الدالة. إليك كيفية إعادة كتابة دالة `updateCounter()` للسماح لها بالوصول إلى متغير `$counter` العام:

```
function updateCounter()
{
 global $counter;
```

```
 $counter++;
}

$counter = 10;
updateCounter();
echo $counter;
11
```

هناك طريقة أكثر تعقيدًا لتحديث المتغير العام وهي استخدام مصفوفة `$GLOBALS` الخاصة بـ PHP بدلاً من الوصول إلى المتغير مباشرةً:

```
function updateCounter()
{
 $GLOBALS['counter']++;
}

$counter = 10;
updateCounter();
echo $counter;
11
```

المتغيرات الثابتة

المتغير الثابت يحتفظ بقيمته بين استدعاءات الدالة ولكنه مرئي فقط داخل تلك الدالة. تقوم بتعريف متغير ثابت باستخدام الكلمة الأساسية `static`. فمثلا:

```
function updateCounter()
{
 static $counter = 0;
```

```
 $counter++;

 echo "Static counter is now {$counter}<br/>";
}

$counter = 10;
updateCounter();
updateCounter();

echo "Global counter is {$counter}";
Static counter is now 1
Static counter is now 2
Global counter is 10
```

معلمات الدوال

كما سنناقش بمزيد من التفصيل في الفصل 3، يمكن أن يكون لتعريف الدالة معلمات مسمى:

```
function greet($name)
{
 echo "Hello, {$name}";
}

greet("Janet");
Hello, Janet
```

معلمات الدالة محلية، مما يعني أنها متاحة فقط داخل دوالها. في هذه الحالة، لا يمكن الوصول إلى `$name` من خارج `greet()`.

جمع القمامة Garbage Collection

تستخدم PHP حساب المراجع *"reference counting"* والنسخ عند الكتابة *"copy-on-write"* لإدارة الذاكرة. يضمن النسخ عند الكتابة عدم إهدار الذاكرة عند نسخ القيم بين المتغيرات، ويضمن حساب المرجع إعادة الذاكرة إلى نظام التشغيل عندما لا تكون هناك حاجة إليها.

لفهم إدارة الذاكرة في PHP، يجب أن تفهم أولاً فكرة جدول الرموز *"symbol table"*. يتكون المتغير من جزأين - اسمه (على سبيل المثال، `$name`) وقيمته (مثل: `"Fred"`). جدول الرموز هو مصفوفة تقوم بتعيين أسماء المتغيرات لمواضع قيمها في الذاكرة.

عندما تنسخ قيمة من متغير إلى آخر، لا تحصل PHP على ذاكرة أكبر لنسخة من القيمة. بدلاً من ذلك، تقوم بتحديث جدول الرموز للإشارة إلى أن "كلا المتغيرين هما اسمان لنفس الجزء من الذاكرة". لذا فإن الكود التالي لا يُنشئ في الواقع مصفوفة جديدة:

```
$worker = array("Fred", 35, "Wilma");
$other = $worker; // array isn't duplicated in memory
```

إذا قمت بتعديل أي من النسختين لاحقًا، فإن PHP تخصص الذاكرة المطلوبة وتقوم بالنسخة:

```
$worker[1] = 36; // array is copied in memory, value changed
```

من خلال تأخير التخصيص والنسخ، توفر PHP الوقت والذاكرة في الكثير من المواقف. هذه نسخة عند الكتابة.

كل قيمة يشير إليها جدول الرموز لها عدد مرجعي *"reference count"*، وهو رقم يمثل عدد الطرق المتاحة للوصول إلى تلك القطعة من الذاكرة. بعد التعيين الأولي للمصفوفة إلى `$worker` و `$worker` إلى `$other`، فإن المصفوفة المشار إليها بإدخالات جدول الرموز لـ `$worker` و `$other` تحتوي على عدد مرجعي يبلغ [1]2. بمعنى آخر، يمكن الوصول إلى هذه الذاكرة بطريقتين: من خلال `$worker` أو `$other`. ولكن بعد تغيير `$worker[1]`، تُنشئ PHP مصفوفة جديدة لـ `$worker`، ويكون عدد المراجع لكل مصفوفة 1 فقط.

عندما يخرج متغير عن النطاق في نهاية دالة، مثل معلمات الوظيفة والمتغيرات المحلية، فإن العدد المرجعي لقيمته ينخفض بمقدار واحد. عندما يتم تعيين قيمة لمتغير في منطقة مختلفة من الذاكرة، يتم تقليل العدد المرجعي للقيمة القديمة بمقدار واحد. عندما يصل العد المرجعي لقيمة إلى 0، يتم تحرير ذاكرتها. هذا هو العد المرجعي.

العد المرجعي هو الطريقة المفضلة لإدارة الذاكرة. احتفظ بالمتغيرات محلية للوظائف، وقم بتمرير القيم التي تحتاج الوظائف للعمل عليها ، ودع حساب المرجع يتولى إدارة الذاكرة. إذا كنت تصر على محاولة الحصول على مزيد من المعلومات أو التحكم في تحرير قيمة المتغير، فاستخدم الدالة `isset()` و `unset()`.

لمعرفة ما إذا تم تعيين متغير على شيء ما - حتى السلسلة الفارغة - استخدم `isset()`:

```
$s1 = isset($name); // $s1 is false
$name = "Fred";
```

```
$s2 = isset($name); // $s2 is true
```

استخدم `unset()` لإزالة قيمة المتغير:

```
$name = "Fred";
unset($name); // $name is NULL
```

التعبيرات والعمليات الرياضية

التعبير "*expression*" هو جزء من كود PHP يمكن تقييمه لإنتاج قيمة. أبسط التعبيرات هي القيم والمتغيرات الحرفية. يتم تقييم القيمة الحرفية لنفسها، بينما يقيم المتغير القيمة المخزنة في المتغير. يمكن تكوين تعبيرات أكثر تعقيدًا باستخدام تعبيرات وعوامل بسيطة.

تأخذ العملية "*operator*" بعض القيم (المعاملات) ويفعل شيئًا (على سبيل المثال، يجمعهم معًا). تتم كتابة العوامل أحيانًا كرموز ترقيم -على سبيل المثال، + و - مألوف لنا من الرياضيات. تقوم بعض العمليات الرياضية بتعديل معاملاتهم، بينما لا يقوم معظمهم بذلك.

يلخص الجدول 2-3 عوامل التشغيل في PHP، والتي تم استعارة العديد منها من C و Perl. يعطي العمود المسمى "P" أسبقية المشغل "operator's precedence"؛ يتم سرد العوامل بترتيب الأسبقية، من الأعلى إلى الأدنى. يعطي العمود المسمى "A" ارتباط المشغل "operator's associativity"، والذي يمكن أن يكون L (من اليسار إلى اليمين)، أو R (من اليمين إلى اليسار)، أو N (غير ارتباط "nonassociative").

الجدول 2-3. عوامل تشغيل PHP

العملية	المشغل أو العامل	A	P
إنشاء كائن جديد	`clone, new`	N	24
Array subscript	[	L	23

P	A	المشغل أو العامل	العملية
22	R	**	الأس
21	R	~	Bitwise Not
	R	++	زيادة
	R	--	نقصان
	R	(int), (bool), (float), (string), (array), (object), (unset)	Cast
	R	@	Inhibit errors
20	N	instanceof	Type testing
19	R	!	ليس (المنطقي)
18	L	*	الضرب
	L	/	القسمة
	L	%	باقي القسمة
17	L	+	الجمع

العملية	المشغل أو العامل	A	P
الناقص	-	L	
تسلسل السلسلة	.	L	
Bitwise shift left	<<	L	16
Bitwise shift right	>>	L	
أقل من، أقل من أو يساوي	<, <=	N	15
أكبر من، أكبر من أو يساوي	>, >=	N	
المساواة في القيمة	==	N	14
عدم المساواة	!=, <>	N	
المساواة في النوع والقيمة	===	N	
لايساوي النوع والقيمة	!==	N	
تُرجع عددًا صحيحًا بناءً على مقارنة معاملين:	<=>	N	

P	A	المشغل أو العامل	العملية
			0 عندما يكون اليسار واليمين متساويين، و 1- عندما يكون اليسار أقل من اليمين، و 1 عندما يكون اليسار أكبر من اليمين.
13	L	&	Bitwise AND
12	L	^	Bitwise XOR
11	L	|	Bitwise OR
10	L	&&	و (المنطقي)
9	L	||	أو (المنطقي)
8	R	??	مقارنة
7	L	?:	عامل شرطي
6	R	=	Assignment
	R	+=, −=, *=, /=, .=, %=, &=, |=, ^=, ~=, <<=, >>=	Assignment with operation

P	A	المشغل أو العامل	العملية
5		`yield from`	العائد من
4		`yield`	العائد
3	L	and	و (المنطقي)
2	L	xor	Logical XOR
1	L	or	أو (المنطقي)

عدد المعاملات

معظم العوامل في PHP هي عوامل ثنائية؛ يقومون بدمج معاملين (أو تعبيرين) في تعبير واحد أكثر تعقيدًا. تدعم PHP أيضًا عددًا من العوامل الأحادية، والتي تحول تعبيرًا واحدًا إلى تعبير أكثر تعقيدًا. أخيرًا، تدعم PHP عددًا قليلاً من العوامل الثلاثية التي تدمج العديد من التعبيرات في تعبير واحد.

أسبقية المعامل

يعتمد الترتيب الذي يتم من خلاله تقييم العوامل في التعبير على أسبقيتها النسبية. على سبيل المثال، قد تكتب:

```
2 + 4 * 3
```

كما ترى في الجدول 2-3، فإن عوامل الضرب والجمع لها أسبقية مختلفة، الضرب أعلى من الجمع. إذن، يتم الضرب قبل عملية الجمع، ويكون الناتج 2 + 12، أو 14 كإجابة. إذا تم عكس أسبقية الجمع والضرب، فستكون الإجابة 6 * 3 أو 18.

لفرض ترتيب معين، يمكنك تجميع المعاملات باستخدام عامل التشغيل المناسب بين قوسين. في مثالنا السابق، للحصول على القيمة 18، يمكنك استخدام هذا التعبير:

```
(2 + 4) * 3
```

من الممكن كتابة جميع التعبيرات المعقدة (التعبيرات التي تحتوي على أكثر من عامل واحد) ببساطة عن طريق وضع المعاملات والعوامل بالترتيب المناسب بحيث تعطي أسبقيتها النسبية الإجابة التي تريدها. ومع ذلك، فإن معظم المبرمجين يكتبون عوامل التشغيل بالترتيب الذي يشعرون أنه أكثر منطقية بالنسبة لهم، ويضيفون أقواسًا للتأكد من أنها منطقية لـ PHP أيضًا. يؤدي الحصول على الأولوية بشكل خاطئ إلى رمز مثل:

```
$x + 2 / $y >= 4 ? $z : $x << $z
```

يصعب قراءة هذا الرمز ومن شبه المؤكد أنه لا يفعل ما توقع المبرمج أن يفعله.

تتمثل إحدى الطرق التي يتعامل بها العديد من المبرمجين مع قواعد الأسبقية المعقدة في لغات البرمجة في تقليل الأسبقية إلى قاعدتين:

- الضرب والقسمة لهما أسبقية أعلى من الجمع والطرح.
- استخدم الأقواس لأي شيء آخر.

عمليات ترابطية Operator Associativity

يحدد الترابط الترتيب الذي يتم فيه تقييم العمليات التي لها نفس ترتيب الأسبقية. على سبيل المثال، انظر إلى:

2 / 2 * 2

عاملي القسمة والضرب لهما نفس الأسبقية، لكن نتيجة التعبير تعتمد على العملية التي نقوم بها أولاً:

```
2 / (2 * 2) // 0.5
(2 / 2) * 2 // 2
```

معاملات القسمة والضرب مترابطة لليسار؛ هذا يعني أنه في حالات الغموض، يتم تقييم العمليات من اليسار إلى اليمين. في هذا المثال، تكون النتيجة الصحيحة هي 2.

التعبئة الضمنية Implicit Casting

العديد من العمليات لديها توقعات بشأن معاملاتهم - على سبيل المثال، تتطلب عمليات الرياضيات الثنائية عادةً أن يكون كلا المعاملين من نفس النوع. يمكن لمتغيرات PHP تخزين الأعداد الصحيحة وأرقام الفاصلة العائمة والسلاسل والمزيد، وللحفاظ على أكبر قدر ممكن من تفاصيل النوع بعيدًا عن المبرمج قدر الإمكان، تحول PHP القيم من نوع إلى آخر حسب الضرورة.

يسمى تحويل قيمة من نوع إلى آخر casting. هذا النوع من التعبئة الضمنية تسمى: التلاعب بالنوع *type juggling* في PHP. يتم عرض قواعد التلاعب بالنوع الذي تقوم به العمليات الحسابية في الجدول 2-4.

الجدول 2-4. قواعد التعبئة الضمنية لعمليات حسابية ثنائية

نوع المعامل الأول	نوع المعامل الثاني	عملية التحويل
عددي صحيح	عشري	يتم تحويل العدد الصحيح إلى رقم عشري
عددي صحيح	نصي	يتم تحويل النص إلى رقم؛ إذا كانت القيمة بعد التحويل هي رقم عشري، يتم تحويل العدد الصحيح إلى رقم عشري.
رقم عشري	نصي	يتم تحويل النص إلى رقم عشري.

بعض العمليات الأخرى لديها توقعات مختلفة لمعاملاتهم، وبالتالي لديهم قواعد مختلفة. على سبيل المثال، يحول عامل تسلسل السلسلة كلا المعاملين إلى سلاسل قبل ربطهما:

```
3 . 2.74 // gives the string 32.74
```

يمكنك استخدام سلسلة في أي مكان نتوقع PHP رقمًا. من المفترض أن تبدأ السلسلة بعدد صحيح أو رقم عشري. إذا لم يتم العثور على رقم في بداية السلسلة، فإن القيمة الرقمية لتلك السلسلة هي 0. إذا كانت السلسلة تحتوي على نقطة (.) أو حرف كبير أو حرف e صغير، فإن تقييمها رقميًا ينتج عنه رقم عشري. فمثلا:

```
"9 Lives" - 1; // 8 (int)
"3.14 Pies" * 2; // 6.28 (float)
"9. Lives" - 1; // 8 (float / double)
"1E3 Points of Light" + 1; // 1001 (float)
```

العمليات الحسابية

العمليات الحسابية هي عمليات ستتعرف عليها من الاستخدام اليومي. معظم العمليات الحسابية ثنائية؛ ومع ذلك، فإن عمليات النفي الحسابي والتأكيد الحسابي أحادية. تتطلب هذه العمليات قيمًا رقمية، ويتم تحويل القيم غير الرقمية إلى قيم رقمية وفقًا للقواعد الموضحة في قسم "عوامل الصب". العوامل الحسابية هي:

الجمع (+):

نتيجة عامل الجمع هو مجموع المعاملين.

الطرح (-):

نتيجة عامل الطرح هو الفرق بين المعاملين -أي قيمة المعامل الثاني مطروح من الأول-.

الضرب (*):

نتيجة عامل الضرب هو حاصل ضرب المعاملين. على سبيل المثال: 3 * 4 هي 12.

القسمة (/):

نتيجة عامل القسمة هي حاصل قسمة المعاملين. يمكن أن تعطي قسمة عددين صحيحين عددًا صحيحًا (على سبيل المثال: 2/4) أو عددا عشريا (على سبيل المثال: 2/1).

باقي القسمة (%):

يحول عامل المقياس كلا المعاملين إلى أعداد صحيحة ويعيد ما تبقى من قسمة المعامل الأول بواسطة المعامل الثاني. على سبيل المثال: 10٪ 6 تعطي الباقي من 4.

النفي الحسابي (-)

يعيد عامل النفي الحسابي المعامل مضروبًا في 1-، مما يؤدي إلى تغيير علامته بشكل فعال. على سبيل المثال: يتم تقييم (4 – 3)- إلى 1. يختلف النفي الحسابي عن عامل الطرح، على الرغم من كتابتهما كعلامة ناقص. النفي الحسابي هو دائمًا أحادي وقبل المعامل. الطرح ثنائي وبين معاملاته.

التوكيد الحسابي (+)

يعرض عامل التوكيد الحسابي المعامل مضروبًا في 1+، والذي ليس له أي تأثير. يتم استخدامه فقط كإشارة بصرية للإشارة إلى علامة القيمة. على سبيل المثال: (3 - 4)+ تُقيم بـ -1، تمامًا كما تفعل (3-4).

الأُس (**)

يُرجع معامل الأُس نتيجة رفع `$var1` إلى الأس `$var2`.

```
$var1 = 5;
$var2 = 3;
echo $var1 ** $var2; // outputs 125
```

عامل تسلسل السلسلة

تعتبر معالجة السلاسل جزءًا أساسيًا من تطبيقات PHP حيث أن PHP لديها عامل تسلسل سلسلة منفصل (.). يلحق عامل التسلسل المعامل الأيمن بالمعامل الأيسر ويعيد السلسلة الناتجة. يتم تحويل المعاملات أولاً إلى سلاسل، إذا لزم الأمر. فمثلا:

```
$n = 5;
$s = 'There were ' . $n . ' ducks.';
// $s is 'There were 5 ducks'
```

عامل التسلسل ذو كفاءة عالية، لأن الكثير من PHP يتلخص في تسلسل السلسلة.

عوامل الزيادة التلقائية والتناقص التلقائي

في البرمجة، تتمثل إحدى العمليات الأكثر شيوعًا في زيادة أو تقليل قيمة متغير بمقدار واحد. توفر عوامل الزيادة التلقائية الأحادية (++) والتناقص التلقائي (--) اختصارات لهذه العمليات الشائعة. هذه العوامل فريدة من نوعها من حيث أنها تعمل فقط على المتغيرات، تغير العمليات قيم معاملاتهم وتعيد قيمة.

هناك طريقتان لاستخدام الزيادة التلقائية أو التناقص التلقائي في التعبيرات. إذا وضعت العامل أمام المعامل، فإنه يُرجع القيمة الجديدة للمعامل (زيادة أو تناقص). إذا وضعت العامل بعد المعامل، فإنه يُرجع القيمة الأصلية للمعامل (قبل الزيادة أو الإنقاص). يسرد الجدول 2-5 العمليات المختلفة.

الجدول 2-5. عمليات الزيادة التلقائية والتناقص التلقائي

العامل	الاسم	القيمة المعادة	التأثير على المتغير
$var++	بعد الزيادة	$var	زيادة
++$var	قبل الزيادة	$var + 1	زيادة

التأثير على المتغير	القيمة المعادة	الاسم	العامل
نقصان	$var	بعد النقصان	$var--
نقصان	$var - 1	قبل النقصان	--$var

يمكن تطبيق هذه العوامل على السلاسل وكذلك الأرقام. إن زيادة الحرف الأبجدي يحوله إلى الحرف التالي في الأبجدية. كما هو موضح في الجدول 2-6، تؤدي زيادة "z" أو "Z" إلى التفافه مرة أخرى إلى "a" أو "A" وزيادة الحرف السابق بمقدار واحد (أو إدراج حرف "a" أو "A" جديد إذا كان في الأول من السلسلة)، كما لو كانت الأحرف في نظام رقم الأساس 26.

الجدول 2-6. زيادة تلقائية بالحروف

يعطيك هذا	زيادة هذا
"b"	"a"
"aa"	"z"
"spba"	"spaz"
"L0"	"K9"
"43"	"42"

عوامل المقارنة

كما يوحي اسمها، يقارن عوامل المقارنة المعاملات. تكون النتيجة دائمًا إما true، إذا كانت المقارنة صحيحة، وfalse على خلاف ذلك.

يمكن أن تكون معاملات عوامل المقارنة على حد سواء رقمية أو سلسلة أو سلسلة واحدة رقمية وسلسلة واحدة. يتحقق العامل من المصداقية بطرق مختلفة قليلاً بناءً على أنواع وقيم المعاملات، إما باستخدام مقارنات رقمية صارمة أو باستخدام مقارنات معجمية (نصية). يوضح الجدول 2-7 متى يتم استخدام كل نوع من الأنواع.

الجدول 2-7. نوع المقارنة التي يتم إجراؤها بواسطة عوامل المقارنة

المقارنة	المعامل الثاني	المعامل الأول
رقمي	ر قم	رقمي
رقمي	سلسلة رقمية بالكامل	سلسلة رقمية بالكامل
رقمي	رقم	سلسلة رقمية بالكامل
معجمي	سلسلة رقمية ليست بالكامل	سلسلة رقمية بالكامل
رقمي	رقم	سلسلة رقمية ليست بالكامل

المقارنة	المعامل الثاني	المعامل الأول
معجمي	سلسلة رقمية ليست بالكامل	سلسلة رقمية ليست بالكامل

شيء واحد مهم يجب ملاحظته هو أنه يتم مقارنة سلسلتين رقميتين كما لو كانتا أرقام. إذا كان لديك سلسلتان تتكونان كليًا من أحرف رقمية وتحتاج إلى مقارنتها معجمًا، فاستخدم الدالة `strcmp()`.

عوامل المقارنة هي:

المساواة (==):

إذا كان كلا المعاملين متساويين، فإن هذا العامل يرجع true، وإلا، فإنها ترجع false.

التطابق (===):

إذا كان كلا المعاملين متساويين وكانا من نفس النوع، فإن هذا العامل يرجع true، وإلا، فإنها ترجع false. لاحظ أن هذا العامل لا يقوم بعملية صب النوع الضمني. يكون هذا العامل مفيدًا عندما لا تعرف ما إذا كانت القيم التي تقارنها من النوع نفسه. قد تتضمن المقارنة البسيطة تحويل القيمة. على سبيل المثال، السلاسل "0.0" و "0" غير متساويتين. يقول العامل == إنهم كذلك، لكن === يقول إنهم ليسوا كذلك.

عدم المساواة (=! أو <>)

إذا كانت المعاملات غير متساوية، فإن هذا العامل يرجع true، وإلا، فإنها ترجع false.

غير متطابق (==!)

إذا كانت المعاملات غير متساوية، أو لم تكن من نفس النوع، فإن هذا العامل يرجع القيمة true، وإلا فإنه يرجع false.

أكبر من (>)

إذا كان المعامل الأيسر أكبر من المعامل الأيمن، فإن هذا العامل يرجع true، وإلا، فإنه يرجع false.

أكبر من أو يساوي (>=)

إذا كان المعامل الأيسر أكبر من أو يساوي المعامل الأيمن، فإن هذا العامل يرجع true، وإلا، فإنه يرجع false.

أقل من (<)

إذا كان المعامل الأيسر أقل من المعامل الأيمن، فإن هذا العامل يرجع true، وإلا، فإنه يرجع false.

أصغر من أو يساوي (<=)

إذا كان المعامل الأيسر أقل من المعامل الأيمن أو مساويًا له، فإن هذا العامل يرجع true، وإلا، فإنه يرجع false.

سفينة الفضاء (<=>)، والمعروفة أيضًا باسم "Darth Vader's TIE Fighter"

عندما يتساوى المعاملين الأيمن والأيسر، فإن هذا العامل يُرجع 0، عندما يكون المعامل الأيسر أقل من المعامل الأيمن، فإنه يُرجع -1، وعندما يكون المعامل الأيسر أكبر من المعامل الأيمن، فإنه يعيد 1.

```
$var1 = 5;
$var2 = 65;

echo $var1 <=> $var2 ; // outputs -1
```

```
echo $var2 <=> $var1 ; // outputs 1
```

عامل دمج فارغ *"Null coalescing operator"* (??)

يقوم هذا العامل بتقييم المعامل الأيمن إذا كان المعامل الأيسر NULL، خلاف ذلك، يتم تقييمه إلى المعامل الأيسر.

```
$var1 = null;
$var2 = 31;

echo $var1 ?? $var2 ; //outputs 31
```

عوامل Bitwise

تعمل عوامل البت على التمثيل الثنائي لمعاملاتها. يتم تحويل كل مُعامل أولاً إلى تمثيل ثنائي للقيمة، كما هو موضح في إدخال عامل النفي على مستوى البت في القائمة التالية. تعمل جميع معاملات البت على الأرقام وكذلك السلاسل، ولكنها تختلف في معالجتها لمعاملات السلسلة ذات الأطوال المختلفة. عوامل تشغيل البت هي:

النفي على مستوى البت *"Bitwise negation"* **(~)**

يتغير عامل النفي على مستوى البت من 1s إلى 0s و 0s إلى 1s في التمثيلات الثنائية للمعاملات. يتم تحويل القيم العشرية إلى أعداد صحيحة قبل حدوث العملية. إذا كان المعامل عبارة عن سلسلة، فإن القيمة الناتجة تكون سلسلة بنفس طول الأصل، مع رفض كل حرف في السلسلة.

أحادي المعامل AND (&)

يقارن العامل AND على مستوى أحادي كل بتة مقابلة في التمثيلات الثنائية للمعاملات. إذا كانت كلتا البتتين 1، فإن البتة المقابلة في النتيجة هي 1؛ خلاف ذلك، فإن البت المقابل هو 0. على سبيل المثال، 0755 و 0671 هو 0651. هذا أسهل قليلاً للفهم إذا نظرنا إلى التمثيل الثنائي. Octal 0755 هو رقم ثنائي 111101101، والثماني 0671 هو رقم ثنائي 110111001. يمكننا بعد ذلك بسهولة رؤية البتات الموجودة في كلا العددين والتوصل إلى الإجابة بصريًا:

```
111101101
& 110111001
  ---------
  110101001
```

الرقم الثنائي 110101001 هو رقم ثماني [2]0651، يمكنك استخدام دالة `bindec()` و `decbin()` و `octdec()` و `decoct()` لتحويل الأرقام ذهابًا وإيابًا عندما تحاول فهم الحساب الثنائي.

إذا كان كلا المعاملين عبارة عن سلاسل، فسيقوم العامل بإرجاع سلسلة يكون فيها كل حرف ناتجًا عن عملية "AND" على مستوى بت بين الحرفين المطابقين في المعاملات. السلسلة الناتجة هي طول أقصر المعاملين؛ يتم تجاهل الأحرف الزائدة في السلسلة الأطول. على سبيل المثال: `"wolf"` و `"cat"` هي `"cad"`.

أحادي المعامل OR (|)

يقارن العامل OR على مستوى أحادي كل بتة مقابلة في التمثيل الثنائي للمعاملات. إذا كانت كلتا البتتين 0، فإن البتة الناتجة تكون 0؛ وبخلاف ذلك، يكون البت الناتج هو 1. على سبيل المثال: 020 | 0755 هو 0775.

إذا كان كلا المعاملين عبارة عن سلاسل، فسيقوم العامل بإرجاع سلسلة يكون فيها كل حرف هو نتيجة عملية "OR" أحاديًا بين الحرفين المتوافقين في المعاملات. السلسلة الناتجة هي طول أطول المعاملين، بينما تكون السلسلة الأقصر مبطنًا في النهاية بـ 0 ثنائي. على سبيل المثال، "pussy" | "cat" هو "suwsy".

Bitwise XOR (^)

يقارن عامل البت XOR كل بتة مقابلة في التمثيل الثنائي للمعاملات. إذا كان أي من البتتين في الزوج، وليس كليهما، هو 1، فإن البتة الناتجة هي 1؛ وبخلاف ذلك، يكون البت الناتج هو 0. على سبيل المثال: 023 ^ 0755 هو 776.

إذا كان كلا المعاملين عبارة عن سلاسل، فسيقوم هذا العامل بإرجاع سلسلة يكون فيها كل حرف ناتجًا عن عملية XOR باتجاه أحادي بين الحرفين المطابقين في المعاملين. إذا كانت الجملتان أطوال مختلفة، فإن السلسلة الناتجة هي طول المعامل الأقصر، ويتم تجاهل الأحرف اللاحقة الإضافية في السلسلة الأطول. على سبيل المثال: "AA" ^ "big drink" تساوي ")#".

التحول الأيسر (<<)

يقوم عامل الإزاحة لليسار بإزاحة البتات في التمثيل الثنائي للمعامل الأيسر إلى اليسار بعدد الأماكن المعطاة في المعامل الأيمن. سيتم تحويل كلا المعاملين إلى أعداد صحيحة إذا لم تكن كذلك بالفعل. يؤدي إزاحة رقم ثنائي إلى اليسار إلى إدخال الرقم 0 باعتباره الجزء الأيمن من الرقم ونقل جميع وحدات البت الأخرى

إلى اليسار في مكان واحد. على سبيل المثال: 1 >> 3 (أو إزاحة 11 ثنائية مكان واحد إلى اليسار) ينتج عنها 6 (ثنائي 110).

لاحظ أن كل مكان على اليسار يتم إزاحة الرقم فيه يؤدي إلى مضاعفة الرقم. نتيجة إزاحة اليسار هي ضرب المعامل الأيسر بمقدار 2 أس المعامل الأيمن.

التحول الأيمن (>>)

يقوم عامل الإزاحة لليمين بإزاحة البتات في التمثيل الثنائي للمعامل الأيسر والمعامل الأيمن بعدد الأماكن المعطاة في المعامل الأيمن. سيتم تحويل كلا المعاملين إلى أعداد صحيحة إذا لم تكن كذلك بالفعل. يؤدي إزاحة رقم ثنائي موجب إلى اليمين إلى إدخال الرقم 0 باعتباره الجزء الأيسر من الرقم ونقل جميع وحدات البت الأخرى إلى المكان الصحيح. يؤدي إزاحة رقم ثنائي سالب إلى اليمين إلى إدخال 1 باعتباره الجزء الأيسر من الرقم ونقل جميع وحدات البت الأخرى إلى اليمين في مكان واحد. يتم تجاهل البت الموجود في أقصى اليمين. على سبيل المثال: 1 >> 3 (أو ثنائي 1101) إزاحة بت واحد إلى اليمين ينتج 6 (ثنائي 110).

العمليات المنطقية

توفر لك العوامل المنطقية طرقًا لبناء تعبيرات منطقية معقدة. تعامل العوامل المنطقية معاملاتها كقيم منطقية وتعيد قيمة منطقية. توجد علامات ترقيم وإصدارات باللغة الإنجليزية من عوامل التشغيل (|| و or هو نفس العامل). العوامل المنطقية هي:

العامل المنطقي (*&&, and*) *AND*

تكون نتيجة عملية AND المنطقية true إذا كان كلا المعاملين true، وإلا فهو false. إذا كانت قيمة المعامل الأول false، فإن العامل المنطقي AND يعرف أن القيمة الناتجة يجب أن تكون false أيضًا، لذلك لا يتم تقييم المعامل الأيمن أبدًا. تسمى هذه العملية بالدائرة القصيرة "*short-circuiting*"، ويستخدمها مصطلح PHP شائع لضمان تقييم جزء من الكود فقط إذا كان هناك شيء صحيح. على سبيل المثال: قد نتصل بقاعدة بيانات فقط إذا كانت بعض العلامات غير false:

```
$result = $flag and mysql_connect();
```

العوامل الـ || و or تختلف فقط في أسبقيتها.

العامل المنطقي (xor) XOR

تكون نتيجة عملية XOR المنطقية true إذا كان أي من المعاملين true وليس كلاهما، وإلا فهو false.

عامل النفي المنطقي (!)

يُرجع عامل النفي المنطقي القيمة المنطقية "true" إذا تم تقييم المعامل إلى "false"، وfalse إذا تم تقييم المعامل إلى "true".

عوامل الصب Casting Operators

على الرغم من أن لغة PHP مكتوبة بشكل ضعيف، إلا أن هناك مناسبات يكون فيها من المفيد اعتبار القيمة كنوع معين. تسمح لك عوامل الصب (int) و (float) و (string) و (bool) و (array) و (object) و (unset) بفرض قيمة في نوع معين. لاستخدام عامل الصب، ضع العامل على يسار المعامل. يسرد الجدول 2-8 عوامل الصب وعوامل التشغيل المترادفة والنوع الذي يغير العامل القيمة إليه.

الجدول 2-8. عوامل صب PHP

تغيير النوع لـ	عوامل مترادفة	العامل
صحيح	(integer)	(int)
منطقي	(boolean)	(bool)
عشري	(double), (real)	(float)
نصي		(string)
مصفوفة		(array)
كائن		(object)
فارغ		(unset)

يؤثر الصب على الطريقة التي يفسر بها العوامل الآخرون القيمة بدلاً من تغيير القيمة في المتغير. على سبيل المثال، الكود:

```
$a = "5";
$b = (int) $a;
```

يعين `$b` القيمة الصحيحة `$a`، يبقى `$a` السلسلة "5". لإعطاء قيمة المتغير نفسه، يجب أن تعيد نتيجة التحويل إلى المتغير:

```
$a = "5";
$a = (int) $a; // now $a holds an integer
```

ليس كل عامل صب مفيد. يعطي إرسال مصفوفة إلى نوع رقمي 1 (إذا كانت المصفوفة فارغة، فإنها تعطي 0)، ويؤدي تحويل المصفوفة إلى سلسلة إلى إعطاء "Array" (رؤية هذا في مخرجاتك هو علامة أكيدة على أنك طبعت متغيرًا يحتوي على مجموعة).

يؤدي إرسال كائن إلى مصفوفة إلى إنشاء مصفوفة من الخصائص، وبالتالي تعيين أسماء الخصائص إلى القيم:

```
class Person
{
 var $name = "Fred";
 var $age = 35;
}

$o = new Person;
$a = (array) $o;
```

```
print_r($a);
Array ( [name] => Fred [age] => 35)
```

يمكنك تحويل مصفوفة إلى كائن لبناء كائن تتوافق خصائصه مع مفاتيح وقيم المصفوفة. فمثلا:

```
$a = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$o = (object) $a;
echo $o->name;
Fred
```

المفاتيح التي لا تعد معرّفات صالحة هي أسماء خصائص غير صالحة ولا يمكن الوصول إليها عندما يتم إرسال مصفوفة إلى كائن، ولكن تتم استعادتها عند إعادة الكائن إلى مصفوفة.

عوامل التخصيص

يقوم عوامل التخصيص بتخزين أو تحديث القيم في المتغيرات. إن عوامل الزيادة التلقائية والتناقص التلقائي التي رأيناها سابقًا هي عوامل تعيين عالية التخصص - وهنا نرى الأشكال الأكثر عمومية. عامل التخصيص الأساسي هو =، لكننا سنرى أيضًا مجموعات من المهام والعمليات الثنائية، مثل: =+ و =&.

التخصيص

عامل التخصيص الأساسي (=) يعين قيمة لمتغير. المعامل الأيسر دائمًا متغير. يمكن أن يكون المعامل الأيمن أي تعبير – أي تعبير حرفي بسيط أو متغير أو معقد. يتم تخزين قيمة المعامل الأيمن في المتغير المسمى بواسطة المعامل الأيسر.

نظرًا لأن جميع العوامل مطلوبة لإرجاع قيمة، فإن عامل التعيين يُرجع القيمة المعينة للمتغير. على سبيل المثال: التعبير $a = 5 لا يعين 5 إلى $a فحسب، بل يتصرف أيضًا كقيمة 5 إذا تم استخدامه في تعبير أكبر. ضع في اعتبارك التعبيرات التالية:

```
$a = 5;
$b = 10;
$c = ($a = $b);
```

يتم تقييم التعبير $a = $b أولاً بسبب الأقواس. الآن، كلا من $a و $b لهما نفس القيمة، 10. أخيرًا، تم تعيين $c نتيجة التعبير $a = $b، وهي القيمة المخصصة للمعامل الأيسر (في هذه الحالة، $a). عندما ينتهي التعبير الكامل من التقييم، تحتوي جميع المتغيرات الثلاثة على نفس القيمة: 10.

التخصيص مع عمليات

بالإضافة إلى عامل التخصيص الأساسي، هناك العديد من عوامل التخصيص الذين يعتبرون اختصارًا مناسبًا. تتكون هذه العوامل من عامل ثنائي متبوعًا مباشرة بعلامة يساوي، ويكون تأثيرها هو نفسه تنفيذ العملية باستخدام المعاملات الكاملة، ثم تعيين القيمة الناتجة إلى المعامل الأيسر. عوامل التخصيص هذه هي:

زائد يساوي (=+)

يضيف المعامل الأيمن إلى قيمة المعامل الأيسر، ثم يخصص النتيجة إلى المعامل الأيسر. `$a += 5` هو نفسه `$a = $a + 5`.

ناقص يساوي (=-)

يطرح المعامل الأيمن من قيمة المعامل الأيسر، ثم يعين النتيجة إلى المعامل الأيسر.

قسمة يساوي (/=)

يقسم قيمة المعامل الأيسر على المعامل الأيمن، ثم يعين النتيجة إلى المعامل الأيسر.

ضرب يساوي (*=)

يضرب المعامل الأيمن في قيمة المعامل الأيسر، ثم يخصص النتيجة للمعامل الأيسر.

المعامل يساوي (%=)

ينفذ عملية المقياس على قيمة المعامل الأيسر والمعامل الأيمن، ثم يخصص النتيجة للمعامل الأيسر.

***Bitwise-XOR-equals* (^=)**

ينفذ XOR باتجاه بت على المعاملين الأيسر والأيمن، ثم يخصص النتيجة للمعامل الأيسر.

***Bitwise-AND-equals* (&=)**

ينفذ "AND" bitwise على قيمة المعامل الأيسر والمعامل الأيمن، ثم يخصص النتيجة إلى المعامل الأيسر.

***Bitwise-OR-equals* (|=)**

ينفذ "OR" باتجاه بت على قيمة المعامل الأيسر والمعامل الأيمن، ثم يعين النتيجة إلى المعامل الأيسر.

تسلسل يساوي (.=)

يربط المعامل الأيمن بقيمة المعامل الأيسر، ثم يخصص النتيجة إلى المعامل الأيسر.

عوامل متنوعة

عوامل تشغيل PHP المتبقية مخصصة لإخماد الأخطاء وتنفيذ أمر خارجي واختيار القيم:

منع الخطأ (@)

يمكن لبعض العوامل أو الدوال إنشاء رسائل خطأ. يتم استخدام عامل منع الأخطاء، الذي تمت مناقشته بالكامل في الفصل 17، لمنع إنشاء هذه الرسائل.

تنفيذ (`...`)

ينفذ عامل backtick السلسلة الموجودة بين backticks كأمر shell ويعيد الإخراج. فمثلا:

```
$listing = `ls -ls /tmp`;
echo $listing;
```

الشرط (: ?)

العامل الشرطي، اعتمادًا على الكود الذي تنظر إليه، هو إما المشغل الأكثر استخدامًا أو الأقل استخدامًا. إنه المشغل الثلاثي الوحيد (ثلاثة معاملات)، وبالتالي يُطلق عليه أحيانًا فقط العامل الثلاثي ternary operator.

يقوم العامل الشرطي بتقييم التعبير قبل؟. إذا كان التعبير صحيحًا، فسيقوم العامل بإرجاع قيمة التعبير بين؟ و:، وبخلاف ذلك، يُرجع العامل قيمة التعبير بعد:. على سبيل المثال:

```
<a href="<? echo $url; ?>"><? echo $linktext ? $linktext :
$url; ?></a>
```

إذا كان نص الرابط $url موجودًا في المتغير $linktext، فسيتم استخدامه كنص للرابط، وإلا، يتم عرض عنوان URL نفسه.

النوع (*instanceof*)

يختبر العامل instanceof ما إذا كان المتغير كائنًا تم إنشاء مثيل له لفئة معينة أو يقوم بتنفيذ واجهة (انظر الفصل 6 لمزيد من المعلومات حول الكائنات والواجهات):

```
$a = new Foo;
$isAFoo = $a instanceof Foo; // true
$isABar = $a instanceof Bar; // false
```

جمل التحكم في التدفق Flow-Control Statements

يدعم PHP عدداً من تركيبات البرمجة التقليدية للتحكم في تدفق تنفيذ البرنامج.

تسمح الجمل الشرطية، مثل: `if/else` and `switch`، للبرنامج بتنفيذ أجزاء مختلفة من التعليمات البرمجية، أو عدم تنفيذ أي منها على الإطلاق، اعتماداً على بعض الشروط. الحلقات، مثل: while و for، تدعم التنفيذ المتكرر لأجزاء معينة من التعليمات البرمجية.

if

تحقق جملة if من مصداقية التعبير، وإذا كان التعبير صحيحاً، يتم تقييم الجملة. تبدو جملة if مثل:

```
if (expression)statement
```

لتحديد جملة بديلة لتنفيذه عندما يكون التعبير خاطئاً، استخدم else:

```
if (expression)
 statement
else statement
```

كمثال:

```
if ($user_validated)
```

```
 echo "Welcome!";
else
 echo "Access Forbidden!";
```

لتضمين أكثر من جملة واحدة في عبارة if، استخدم block - مجموعة من الجمل محاطة بأقواس معقوفة:

```
if ($user_validated) {
 echo "Welcome!";
 $greeted = 1;
}
else {
 echo "Access Forbidden!";
 exit;
}
```

توفر PHP صيغة أخرى للكتل في الاختبارات والحلقات. بدلاً من إرفاق كتلة الجمل بأقواس معقوفة، قم بإنهاء سطر if بنقطتين (:) واستخدم كلمة رئيسية محددة لإنهاء الكتلة (endif، في هذه الحالة). فمثلا:

```
if ($user_validated):
 echo "Welcome!";
 $greeted = 1;
else:
 echo "Access Forbidden!";
 exit;
endif;
```

الجمل الأخرى الموضحة في هذا الفصل لها أيضًا أنماط بناء جملة بديلة مماثلة (وكلمات رئيسية نهائية)؛ يمكن أن تكون مفيدة إذا كان لديك كتل كبيرة من HTML داخل جملك. فمثلا:

```
<?php if ($user_validated) : ?>
 <table>
 <tr>
 <td>First Name:</td><td>Sophia</td>
 </tr>
 <tr>
 <td>Last Name:</td><td>Lee</td>
 </tr>
 </table>
<?php else: ?>
 Please log in.
<?php endif ?>
```

لأنه إذا كانت جملة، فيمكنك ربط (تضمين) أكثر من جملة. هذا أيضًا مثال جيد على كيفية استخدام الكتل للمساعدة في الحفاظ على تنظيم الأشياء:

```
if ($good) {
 print("Dandy!");
}
else {
 if ($error) {
 print("Oh, no!");
 }
 else {
 print("I'm ambivalent...");
 }
```

```
}
```

هذه السلاسل من جمل if شائعة بدرجة كافية بحيث توفر PHP صياغة أسهل: جملة elseif. على سبيل المثال، يمكن إعادة كتابة الكود السابق على النحو التالي:

```
if ($good) {
 print("Dandy!");
}
elseif ($error) {
 print("Oh, no!");
}
else {
 print("I'm ambivalent...");
}
```

يمكن استخدام المعامل الشرطي الثلاثي (: ?) لتقصير اختبارات الصواب / الخطأ البسيطة. اتخذ موقفًا شائعًا، مثل التحقق لمعرفة ما إذا كان متغير معين صحيحًا وطباعة شيء ما إذا كان كذلك. مع جملة if/else العادية، يبدو الأمر كما يلي:

```
<td><?php if($active) { echo "yes"; } else { echo "no"; } ?></td>
```

مع عامل التشغيل الشرطي الثلاثي، يبدو الأمر كما يلي:

```
<td><?php echo $active ? "yes" : "no"; ?></td>
```

مقارنة بنية الجملة للاثنين:

if (*expression*) { *true_statement* } **else** { *false_statement* }

$(expression) \ ? \ true_expression : false_expression$

الاختلاف الرئيسي هنا هو أن العامل الشرطي ليس جملة على الإطلاق. هذا يعني أنه يتم استخدامه في التعبيرات، ونتيجة التعبير الثلاثي الكامل هو في حد ذاته تعبير. في المثال السابق، تكون جملة echo داخل شرط if، بينما عند استخدامها مع العامل الثلاثي، فإنها تسبق التعبير.

switch

قد تحدد قيمة متغير واحد واحدًا من عدد من الخيارات المختلفة (على سبيل المثال، يحتفظ المتغير باسم المستخدم وتريد أن تفعل شيئًا مختلفًا لكل مستخدم). تم تصميم بيان التبديل لهذا الموقف فقط.

تُعطى جملة switch تعبيرًا ويقارن قيمته بجميع الحالات في الـ switch؛ يتم تنفيذ جميع الجمل في حالة المطابقة، حتى أول كلمة رئيسية break تعثر عليها. إذا لم يكن هناك تطابق، وتم تحديد القيمة `default`، فسيتم تنفيذ جميع الجمل التي تتبع الكلمة الأساسية `default`، حتى أول كلمة رئيسية break مصادفة.

على سبيل المثال، افترض أن لديك ما يلي:

```
if ($name == 'ktatroe') {
 // do something
}
else if ($name == 'dawn') {
 // do something
```

```
}
else if ($name == 'petermac') {
 // do something
}
else if ($name == 'bobk') {
 // do something
}
```

يمكنك استبدال هذه الجملة بجملة التبديل التالية:

```
switch($name) {
 case 'ktatroe':
 // do something
 break;
 case 'dawn':
 // do something
 break;
 case 'petermac':
 // do something
 break;
 case 'bobk':
 // do something
 break;
}
```

الصيغة البديلة لذلك هي:

```
switch($name):
 case 'ktatroe':
```

```
 // do something
 break;
 case 'dawn':
 // do something
 break;
 case 'petermac':
 // do something
 break;
 case 'bobk':
 // do something
 break;
endswitch;
```

نظراً لأنه يتم تنفيذ الجمل من تسمية حالة المطابقة إلى الكلمة الأساسية `break` التالية، يمكنك دمج العديد من الحالات في *fall-through*. في المثال التالي، تتم طباعة "yes" عندما يكون $name مساويًا لـ `sylvie` أو `bruno`:

```
switch ($name) {
 case 'sylvie': // fall-through
 case 'bruno':
 print("yes");
 break;
 default:
 print("no");
 break;
}
```

يعد التعليق على حقيقة أنك تستخدم حالة سقوط fall-through في التبديل فكرة جيدة، لذلك لا يأتي شخص ما في وقت ما ويضيف `break` معتقدًا أنك قد نسيتها.

يمكنك تحديد عدد اختياري من المستويات للكلمات الرئيسية `break`. بهذه الطريقة، يمكن أن تخرج جملة break من عدة مستويات من جمل التبديل `switch` المتداخلة. يتم عرض مثال على استخدام break بهذه الطريقة في القسم التالي.

while

أبسط شكل من أشكال الحلقة هو جملة while:

```
while (expression)statement
```

إذا تم تقييم التعبير "expression" إلى true، فسيتم تنفيذ الجملة "statement" ثم إعادة تقييم التعبير (إذا كان لا يزال صحيحًا، فسيتم تنفيذ جسم الحلقة مرة أخرى، وهكذا). تخرج الحلقة عندما لا يكون التعبير صحيحًا (أي يتم تقييمه على false).

على سبيل المثال، إليك بعض التعليمات البرمجية التي تضيف الأعداد الصحيحة من 1 إلى 10:

```
$total = 0;
$i = 1;

while ($i <= 10) {
 $total += $i;
 $i++;
}
```

الصيغة البديلة لـ while لها هذه البنية:

```
while (expr):
 statement;
 more statements ;
endwhile;
```

كمثال:

```
$total = 0;
$i = 1;

while ($i <= 10):
 $total += $i;
 $i++;
endwhile;
```

يمكنك الخروج من حلقة قبل الأوان باستخدام الكلمة الأساسية `break`. في الكود التالي، `$i` لا تصل أبدًا إلى القيمة 6؛ لأن الحلقة تتوقف بمجرد وصولها إلى 5:

```
$total = 0;
$i = 1;

while ($i <= 10) {
 if ($i == 5) {
 break; // breaks out of the loop
 }
```

```
 $total += $i;
 $i++;
}
```

اختياريًا، يمكنك وضع رقم بعد الكلمة الأساسية break يشير إلى عدد مستويات هيا كل الحلقة التي يجب الخروج منها. بهذه الطريقة، يمكن أن تندلع عبارة مدفونة عميقاً في حلقات متداخلة من الحلقة الخارجية. فمثلا:

```
$i = 0;
$j = 0;

while ($i < 10) {
 while ($j < 10) {
 if ($j == 5) {
 break 2; // breaks out of two while loops
 }

 $j++;
 }

 $i++;
}

echo "{$i}, {$j}";
```

0, 5

تتخطى جملة continue إلى الاختبار التالي لحالة الحلقة. كما هو الحال مع الكلمة الأساسية break، يمكنك المتابعة عبر عدد اختياري من مستويات بنية الحلقة:

```
while ($i < 10) {
 $i++;

 while ($j < 10) {
 if ($j == 5) {
 continue 2; // continues through two levels
 }

 $j++;
 }
}
```

في هذا الكود، لا يحتوي $j على قيمة أعلى من 5، ولكن $i يمر عبر جميع القيم من 0 إلى 9.

تدعم PHP أيضًا حلقة do/while، والتي تأخذ الشكل التالي:

```
do
 statement
while (expression)
```

استخدم حلقة do/while لضمان تنفيذ جسم الحلقة مرة واحدة على الأقل (المرة الأولى):

```
$total = 0;
$i = 1;
```

```
do {
 $total += $i++;
} while ($i <= 10);
```

يمكنك استخدام جمل break و continue في جمل do / while تمامًا كما هو الحال في جملة while العادية.

تُستخدم جملة do/while أحيانًا لكسر كتلة التعليمات البرمجية عند حدوث حالة خطأ. فمثلا:

```
do {
 // do some stuff

 if ($errorCondition) {
 break;
 }

 // do some other stuff
} while (false);
```

لأن شرط الحلقة خاطئ، يتم تنفيذ الحلقة مرة واحدة فقط، بغض النظر عما يحدث داخل الحلقة. ومع ذلك، في حالة حدوث خطأ، لا يتم تقييم الكود بعد الفاصل.

for

الجملة for مشابهة لجملة while، إلا أنها تضيف عدادات "counter initialization" وتعبيرات عداد "counter manipulation expressions"، وغالبًا ما تكون أقصر وأسهل في القراءة من جمل حلقة while.

إليك حلقة while التي تعد من 0 إلى 9، وطباعة كل رقم:

```
$counter = 0;

while ($counter < 10) {
 echo "Counter is {$counter} <br/>";
 $counter++;
}
```

إليك حلقة for المقابلة والأكثر إيجازًا:

```
for ($counter = 0; $counter < 10; $counter++) {
 echo "Counter is $counter <br/>";
}
```

بنية الجملة for هو:

```
for (start; condition; increment) { statement(s); }
```

يتم تقييم بداية "start" التعبير "expression" مرة واحدة، في بداية جملة for. في كل مرة خلال الحلقة، يتم اختبار شرط "*condition*" التعبير. إذا كان هذا صحيحًا، فسيتم تنفيذ جسم الحلقة؛ إذا كانت خاطئة، تنتهي الحلقة. يتم تقييم زيادة "*increment*" التعبير بعد تشغيل جسم الحلقة.

الصيغة البديلة لجملة for هي:

```
for (expr1; expr2; expr3):
 statement;
 ...;
endfor;
```

يضيف هذا البرنامج الأرقام من 1 إلى 10 باستخدام حلقة for:

```
$total = 0;

for ($i= 1; $i <= 10; $i++) {
 $total += $i;
}
```

إليك نفس الحلقة باستخدام البنية البديلة:

```
$total = 0;

for ($i = 1; $i <= 10; $i++):
 $total += $i;
endfor;
```

يمكنك تحديد عدة تعبيرات لأي من التعبيرات في جملة for عن طريق فصل التعبيرات بفاصلات. فمثلا:

```
$total = 0;

for ($i = 0, $j = 1; $i <= 10; $i++, $j *= 2) {
```

```
 $total += $j;
}
```

يمكنك أيضًا ترك تعبير فارغ، مما يشير إلى أنه لا ينبغي فعل أي شيء لهذه المرحلة. في الشكل الأكثر انحطاطًا، تصبح جملة for حلقة لا نهائية. ربما لا ترغب في تشغيل هذا المثال، لأنه لا يتوقف عن الطباعة:

```
for (;;) {
 echo "Can't stop me!<br />";
}
```

في حلقات for، كما هو الحال في حلقات while، يمكنك استخدام الكلمات الرئيسية break وcontinue لإنهاء الحلقة أو التكرار الحالي.

foreach

تسمح لك جملة foreach بتكرار العناصر في المصفوفة. تمت مناقشة شكلي جملة foreach بمزيد من التفصيل في الفصل الخامس، حيث نتحدث بمزيد من العمق عن المصفوفات. للتكرار على مصفوفة للوصول إلى القيمة عند كل مفتاح، استخدم:

```
foreach ($array as $current) {
 // ...
}
```

الصيغة البديلة هي:

```
foreach ($array as $current):
 // ...
endforeach;
```

للتكرار على مصفوفة للوصول إلى كل من المفتاح والقيمة، استخدم:

```
foreach ($array as $key => $value) {
 // ...
}
```

الصيغة البديلة هي:

```
foreach ($array as $key => $value):
 // ...
endforeach;
```

try...catch

إن بنية try ... catch ليست بنية تحكم في التدفق بقدر ما هي طريقة أكثر رشاقة للتعامل مع أخطاء النظام. على سبيل المثال، إذا كنت تريد التأكد من أن تطبيق الويب الخاص بك لديه اتصال صالح بقاعدة بيانات قبل المتابعة، فيمكنك كتابة كود مثل هذا:

```
try {
 $dbhandle = new PDO('mysql:host=localhost; dbname=library', $username, $pwd);
 doDB_Work($dbhandle); // call function on gaining a connection
 $dbhandle = null; // release handle when done
}
catch (PDOException $error) {
 print "Error!: " . $error->getMessage() . "<br/>";
 die();
}
```

هنا تتم محاولة الاتصال في جزء try من البناء وإذا كان هناك أي أخطاء به، فإن تدفق الكود يقع تلقائيًا في جزء catch، حيث يتم إنشاء مثيل فئة PDOException في متغير $error. يمكن بعد ذلك عرضها على الشاشة ويمكن أن يفشل الكود "بأمان"، بدلاً من وضع نهاية مفاجئة. يمكنك حتى إعادة التوجيه لمحاولة الاتصال بقاعدة بيانات بديلة، أو الرد على الخطأ بأي طريقة أخرى تريدها داخل جزء catch.

ملاحظة:

راجع الفصل 9 لمزيد من الأمثلة عن catch ... try فيما يتعلق بـ PDO (كائنات بيانات PHP "PHP Data Objects") ومعالجة المعاملات.

الإعلان declare

جملة declare تسمح لك بتحديد توجيهات التنفيذ لكتلة من التعليمات البرمجية. بنية جملة declare هو:

```
declare (directive)statement
```

حاليًا، لا يوجد سوى ثلاثة أشكال لـ declare: الـ ticks، والـ encoding، والـ strict_types. يمكنك استخدام توجيه ticks لتحديد عدد المرات (التي يتم قياسها تقريبًا في عدد جمل الكود) يتم تسجيل دالة ticks عند استدعاء register_tick_function(). فمثلا:

```
register_tick_function("someFunction");

declare(ticks = 3) {
 for($i = 0; $i < 10; $i++) {
```

```
 // do something
 }
}
```

في هذا الكود، يتم استدعاء `someFunction()` بعد تنفيذ كل جملة ثالثة داخل الكتلة.

يمكنك استخدام توجيه `encoding` لتحديد ترميز إخراج نص PHP البرمجي. فمثلا:

```
declare(encoding = "UTF-8");
```

يتم تجاهل هذا النموذج من جملة `declare` إلا إذا جمعت PHP باستخدام الخيار `--enable-zend-multibyte`.

أخيرًا، يمكنك استخدام التوجيه `strict_types` لفرض استخدام أنواع البيانات الصارمة عند تعريف المتغيرات واستخدامها.

exit and return

بمجرد الوصول إليها، تنهي جملة exit تنفيذ البرنامج النصي. ترجع جملة return من دالة أو من البرنامج النصي في المستوى الأعلى من البرنامج.

جملة exit تأخذ قيمة اختيارية. إذا كان هذا رقمًا، فهو يمثل حالة الخروج من العملية. إذا كانت سلسلة، تتم طباعة القيمة قبل إنهاء العملية. الدالة `die()` هي اسم مستعار لهذا الشكل من جملة exit:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD);

if (!$db) {
 die("Could not connect to database");
}
```

يتم كتابة هذا بشكل أكثر شيوعًا على النحو التالي:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD)
 or die("Could not connect to database");
```

راجع الفصل 3 لمزيد من المعلومات حول استخدام جملة الإرجاع في الوظائف.

goto

تسمح جملة goto للتنفيذ "بالقفز" إلى مكان آخر في البرنامج. يمكنك تحديد نقاط التنفيذ عن طريق إضافة تسمية "label"، وهي عبارة عن معرف متبوع بنقطتين (:). ثم تنتقل بعد ذلك إلى label من موقع آخر في البرنامج النصي عبر جملة goto:

```
for ($i = 0; $i < $count; $i++) {
 // oops, found an error
 if ($error) {
 goto cleanup;
 }
}

cleanup:
// do some cleanup
```

يمكنك فقط الحصول على label في نفس النطاق مثل جملة goto نفسها، ولا يمكنك القفز إلى حلقة أو تبديل. بشكل عام، في أي مكان قد تستخدم فيه goto (أو جملة break، لهذا الأمر)، يمكنك إعادة كتابة الكود ليكون أكثر نظافة بدونه.

تضمين كود

توفر PHP بنائين لتحميل التعليمات البرمجية و HTML من وحدة نمطية أخرى: `require` و `include`. كلاهما يقوم بتحميل ملف أثناء تشغيل برنامج PHP نصي، ويعملان في الشرطية والحلقات، ويشكو إذا تعذر العثور على الملف الجاري تحميله. توجد الملفات بواسطة مسار ملف مضمن كجزء من التوجيه في استخدام الدالة، أو بناءً على إعداد `include_path` في ملف *php.ini*. يمكن تجاوز طريقة `include_path` بواسطة الدالة `set_include_path()`. إذا فشلت كل هذه الطرق، فإن محاولة PHP الأخيرة هي محاولة العثور على الملف في نفس المجلد مثل البرنامج النصي للاستدعاء. الاختلاف الرئيسي هو أن في `require` إذا طلبت ملف غير موجود يظهر fatal error، بينما محاولة تضمين "`include`" مثل هذا الملف تنتج تحذيرًا ولكنها لا توقف تنفيذ البرنامج النصي.

الاستخدام الشائع لـ `include` هو فصل المحتوى الخاص بالصفحة عن تصميم الموقع العام. تنتقل العناصر الشائعة مثل الرؤوس والتذييلات في ملفات HTML منفصلة، وبعد ذلك تبدو كل صفحة كما يلي:

```
<?php include "header.html"; ?>
content
<?php include "footer.html"; ?>
```

نستخدم التضمين "include" لأنه يسمح لـ PHP بمواصلة معالجة الصفحة حتى إذا كان هناك خطأ في ملف (ملفات) تصميم الموقع. الـ require أقل تسامحًا وتكون أكثر ملاءمة لتحميل مكتبات التعليمات البرمجية، حيث لا يمكن عرض الصفحة إذا لم يتم تحميل المكتبات. فمثلا:

```
require "codelib.php";
mysub(); // defined in codelib.php
```

هناك طريقة أكثر فاعلية للتعامل مع الرؤوس والتذييلات وهي تحميل ملف واحد ثم استدعاء الدوال لإنشاء عناصر الموقع الموحدة:

```
<?php require "design.php";
header(); ?>
content
<?php footer();
```

إذا لم تستطع PHP تحليل جزء من الملف المضاف بواسطة التضمين "include" أو الطلب "require"، تتم طباعة تحذير ويستمر التنفيذ. يمكنك كتم صوت التحذير عن طريق تقديم المكالمة مسبقًا باستخدام عامل الصمت (@) - على سبيل المثال، @include.

إذا تم تمكين خيار allow_url_fopen من خلال ملف تكوين PHP، php.ini، فيمكنك تضمين ملفات من موقع بعيد عن طريق توفير عنوان URL بدلاً من مسار محلي بسيط:

```
include "http://www.example.com/codelib.php";
```

إذا كان اسم الملف يبدأ بـ *http://* أو *https://* أو *ftp://*، فسيتم استرداد الملف من موقع بعيد وتحميله.

يمكن تسمية الملفات المضمنة بـ `include` و`require` بشكل تعسفي. الامتدادات الشائعة هي:
.php, *.php*5, and *.html*.

ملاحظة:

لاحظ أن إحضار ملف ينتهي بـ *php.* عن بُعد من خادم ويب تم تمكين PHP له يؤدي إلى جلب نص *output* PHP البرمجي هذا — إنه ينفذ كود PHP في هذا الملف.

إذا كان أحد البرامج يستخدم `include` أو `require` لتضمين نفس الملف مرتين (تم القيام به عن طريق الخطأ في حلقة، على سبيل المثال)، يتم تحميل الملف وتشغيل الكود، أو طباعة HTML مرتين. يمكن أن يؤدي هذا إلى أخطاء حول إعادة تعريف الوظائف، أو إرسال نسخ متعددة من الرؤوس أو HTML، لمنع حدوث هذه الأخطاء، استخدم التركيبات `include_once` و `require_once`. يتصرفون بنفس سلوك `include` و`require` المرة الأولى التي يتم فيها تحميل الملف، لكنهم يتجاهلون بهدوء المحاولات اللاحقة لتحميل نفس الملف. على سبيل المثال، تحتاج العديد من عناصر الصفحة، كل واحدة مخزنة في ملفات منفصلة، إلى معرفة تفضيلات المستخدم الحالي. يجب أن تقوم مكتبات العناصر بتحميل مكتبة تفضيلات المستخدم مع `require_once`. يمكن لمصمم الصفحة بعد ذلك تضمين عنصر صفحة دون القلق بشأن ما إذا كان قد تم تحميل كود تفضيل المستخدم بالفعل.

يتم استيراد الكود الموجود في ملف مضمن في النطاق الذي يتم فيه العثور على جملة `include`، لذلك يمكن للكود المضمن رؤية متغيرات التعليمات البرمجية الخاصة بك وتغييرها. يمكن أن يكون هذا مفيدًا - على سبيل المثال، قد تخزن مكتبة تتبع المستخدم اسم المستخدم الحالي في المتغير `$user` العام:

```
// main page
include "userprefs.php";
echo "Hello, {$user}.";
```

يمكن أن تكون قدرة المكتبات على رؤية المتغيرات الخاصة بك وتغييرها مشكلة أيضًا. يجب أن تعرف كل متغير عام تستخدمه مكتبة للتأكد من أنك لا تحاول عن طريق الخطأ استخدام أحدها لأغراضك الخاصة، وبالتالي الكتابة فوق قيمة المكتبة وتعطيل طريقة عملها.

إذا كانت بنية `include` أو `require` في دالة، فإن المتغيرات في الملف المضمن تصبح متغيرات نطاق الدالة لتلك الدالة.

نظرًا لأن الكلمات الرئيسية `include` و `require`، وليست جمل حقيقية، فيجب عليك دائمًا تضمينها بأقواس معقوفة في الجمل الشرطية والحلقة:

```
for ($i = 0; $i < 10; $i++) {
 include "repeated_element.html";
}
```

استخدم دالة `get_included_files()` لمعرفة الملفات التي تم تضمينها أو طلبها في النص البرمجي. تقوم بإرجاع مصفوفة تحتوي على أسماء ملفات مسار النظام الكاملة لكل ملف مضمن أو مطلوب. لا يتم تضمين الملفات التي لم يتم تحليلها في هذه المصفوفة.

تضمين PHP في صفحات الويب

على الرغم من أنه من الممكن كتابة وتشغيل برامج PHP مستقلة، فإن معظم أكواد PHP مضمنة في ملفات HTML أو XML. هذا، بعد كل شيء، هو سبب إنشائها في المقام الأول. تتضمن معالجة مثل هذه المستندات استبدال كل جزء من كود مصدر PHP بالمخرجات التي تنتجها عند تنفيذها.

نظراً لأن الملف الفردي عادةً ما يحتوي على PHP وكود مصدر غير PHP، فنحن بحاجة إلى طريقة لتحديد مناطق كود PHP المراد تنفيذه. توفر PHP أربع طرق مختلفة للقيام بذلك.

كما سترى، الطريقة الأولى والمفضلة تبدو مثل XML. الطريقة الثانية تبدو مثل SGML. الطريقة الثالثة تعتمد على أوسمة ASP. الطريقة الرابعة تستخدم وسم HTML القياسية <script>؛ هذا يجعل من السهل تحرير الصفحات باستخدام PHP الممكّن باستخدام محرر HTML عادي.

نمط قياسي (XML)

نظراً لظهور لغة التوصيف القابلة للتوسيع "eXtensible Markup Language" (XML) وترحيل HTML إلى لغة (XHTML) XML، فإن الأسلوب المفضل حالياً لتضمين PHP يستخدم علامات متوافقة مع XML للإشارة إلى تعليمات PHP.

كان من السهل إنشاء أوسمة لتعيين أوامر PHP في XML؛ لأن XML يسمح بتعريف الأوسمة الجديدة. لاستخدام هذا النمط، قم بإحاطة كود PHP بـ ?> and <?php. يتم تفسير كل شيء بين هذه الأوسمة على

أنه PHP، ولا يتم تفسير أي شيء خارج الأوسمة. على الرغم من أنه ليس من الضروري تضمين مسافات بين العلامات والنص المرفق، فإن القيام بذلك يحسن إمكانية القراءة. على سبيل المثال، لجعل PHP تطبع ”Hello, world,“، يمكنك إدراج السطر التالي في صفحة الويب:

```
<?php echo "Hello, world"; ?>
```

الفاصلة المنقوطة اللاحقة في الجملة اختيارية، لأن نهاية الكتلة تفرض أيضًا نهاية الجملة. مضمن في ملف HTML كامل، يبدو هذا كما يلي:

```
<!doctype html>
<html>
<head>
 <title>This is my first PHP program!</title>
</head>

<body>
<p>
 Look, ma! It's my first PHP program:<br />
 <?php echo "Hello, world"; ?><br />
 How cool is that?
</p>
</body>

</html>
```

بالطبع، هذا ليس مثيرًا للغاية - كان بإمكاننا فعل ذلك بدون PHP. تأتي القيمة الحقيقية لـ PHP عندما نضع معلومات حيوية من مصادر مثل قواعد البيانات وقيم النموذج في صفحة الويب. هذا لفصل لاحق،

مع ذلك، دعنا نعود إلى مثالنا "Hello, world". عندما يزور مستخدم هذه الصفحة ويعرض مصدرها، يبدو الأمر كما يلي:

```
<!doctype html>
<html>
<head>
 <title> This is my first PHP program! </title>
</head>

<body>
<p>
 Look, ma! It's my first PHP program:<br />
 Hello, world!<br />
 How cool is that?
</p>
</body>

</html>
```

لاحظ أنه لا يوجد أي أثر لكود مصدر PHP من الملف الأصلي. يرى المستخدم فقط ناتجها.

لاحظ أيضًا أننا قمنا بالتبديل بين PHP و non-PHP، كل ذلك في مساحة سطر واحد. يمكن وضع تعليمات PHP في أي مكان في الملف، حتى ضمن أوسمة HTML الصالحة. فمثلا:

```
<input type="text" name="first_name" value="<?php echo "Peter";
?>" />
```

عندما ينتهي PHP من هذا النص، سيقرأ:

```
<input type="text" name="first_name" value="Peter" />
```

لا يجب أن يكون كود PHP داخل علامات الفتح والإغلاق على نفس السطر. إذا كانت علامة إغلاق تعليمة PHP هي آخر شيء في السطر، فسيتم أيضًا إزالة فاصل السطر الذي يلي علامة الإغلاق. وبالتالي، يمكننا استبدال تعليمات PHP في مثال "Hello, world" بـ:

```
<?php
echo "Hello, world"; ?>
<br />
```

مع عدم وجود تغيير في HTML الناتج.

نمط SGML

يأتي نمط آخر لتضمين PHP من أوسمة معالجة تعليمات SGML. لاستخدام هذه الطريقة، ضع PHP داخل `<?` و`?>`. إليك مثال "Hello, world" مرة أخرى:

```
<? echo "Hello, world"; ?>
```

هذا النمط، المعروف باسم الأوسمة القصيرة *short tags*، مغلق افتراضياً. يمكنك تشغيل دعم الأوسمة القصيرة عن طريق إنشاء PHP باستخدام الخيار `--enable-short-tags`، أو تمكين `short_open_tag` في ملف تكوين PHP. هذا غير محبذ لأنه يعتمد على حالة هذا الإعداد، إذا قمت بتصدير الكود الخاص بك إلى نظام أساسي آخر، فقد يعمل وقد لا يعمل.

وسم `echo` قصير، `<?= ... ?>`، متاحة بغض النظر عن توفر الأوسمة القصيرة.

echo المحتوى مباشرة

ربما تكون العملية الفردية الأكثر شيوعًا داخل تطبيق PHP هي عرض البيانات للمستخدم. في سياق تطبيق الويب، يعني هذا إدخال معلومات مستند HTML التي ستصبح HTML عند عرضها من قبل المستخدم.

لتبسيط هذه العملية، توفر PHP إصدارًا خاصًا من أوسمة SGML التي تأخذ القيمة تلقائيًا داخل الوسم وتدرجها في صفحة HTML. لاستخدام هذه الميزة، أضف علامة يساوي (=) إلى وسم الفتح. باستخدام هذه التقنية، يمكننا إعادة كتابة مثال النموذج الخاص بنا على النحو التالي:

```
<input type="text" name="first_name" value="<?= "Dawn"; ?>">
```

ماذا بعد

الآن بعد أن أصبح لديك أساسيات اللغة تحت حزامك - فهم أساسي لماهية المتغيرات وكيفية تسميتها وأنواع البيانات وكيف يعمل التحكم في تدفق التعليمات البرمجية - سننتقل إلى بعض التفاصيل الدقيقة للغة PHP . سنغطي بعد ذلك ثلاثة موضوعات مهمة جدًا لـ PHP بحيث يكون لكل منها فصوله المخصصة: كيفية تحديد الدوال (الفصل 3)، والتعامل مع السلاسل (الفصل 4)، وإدارة المصفوفات (الفصل 5).

(1) إنها في الواقع 3 إذا كنت تنظر إلى عدد المراجع من C API، ولكن لأغراض هذا الشرح ومن منظور مساحة المستخدم ، فمن الأسهل التفكير فيه على أنه 2.

(2) إليك نصيحة: قسّم الرقم الثنائي إلى ثلاث مجموعات - 6 عبارة عن ثنائي 110، و 5 هو ثنائي 101، و 1 هو ثنائي 001؛ وبالتالي، 0651 هو 110101001.

الفصل الثالث الدوال

الدالة: عبارة عن كتلة مسماة من التعليمات البرمجية تؤدي مهمة محددة، وربما تعمل على مجموعة من القيم المعطاة لها، تعرف أيضًا باسم المعلمات "*parameters*"، وربما تعيد قيمة واحدة أو مجموعة من القيم في مصفوفة. توفر الدوال وقت الترجمة - بغض النظر عن عدد المرات التي تستدعى فيها، يتم تجميع الدوال مرة واحدة فقط للصفحة. كما أنها تعمل على تحسين الموثوقية من خلال السماح لك بإصلاح أي أخطاء في مكان واحد بدلاً من أي مكان تقوم فيه بمهمة ما، كما تعمل على تحسين إمكانية القراءة عن طريق عزل التعليمات البرمجية التي تؤدي مهام محددة.

يقدم هذا الفصل بناء جملة استدعاءات الدوال وتعريفات الدوال ويناقش كيفية إدارة المتغيرات في الدوال وتمرير القيم إليها (بما في ذلك التمرير بالقيمة والمرجع التمريري "including pass-by-value and pass-by-reference"). كما أنه يغطي الدوال المتغيرة والدوال المجهولة.

استدعاء الدالة

يمكن دمج الدوال في برنامج PHP (أو من خلال كونها في ملحق، مضمنة بشكل فعال) أو معرفة من قبل المستخدم. بغض النظر عن مصدرها، يتم تقييم جميع الدوال بنفس الطريقة:

$someValue = *function_name*([*parameter, ...*]);

يختلف عدد المعلمات التي تتطلبها الدالة من دالة إلى أخرى (وكما سنرى لاحقًا، قد تختلف أيضًا لنفس الدالة). قد تكون المعلمات التي يتم توفيرها للدالة أي تعبير صالح ويجب أن تكون بالترتيب المحدد الذي

نتوقعه الدالة. إذا تم إعطاء المعلمات خارج الترتيب، فقد يستمر تشغيل الدالة بواسطة fluke، ولكنها في الأساس حالة "garbage in = garbage out". ستخبرك وثائق الدالة بالمعلمات التي نتوقعها الدالة والقيمة (القيم) التي يمكنك توقع إرجاعها.

فيما يلي بعض الأمثلة على الدوال:

```
// strlen() هي دالة مضمنة ترجع طول السلسلة
$length = strlen("PHP"); // $length الان 3
// sin() و asin() هي دوال رياضية الجيب والقوس
$result = sin(asin(1)); // $result is the sine of arcsin(1), or
1.0

// unlink() تحذف ملف
$result = unlink("functions.txt");
// $result = true or false depending on success or failure
```

في المثال الأول، نعطي المدخل "PHP" للدالة `strlen()` والتي تعطينا عدد الأحرف في السلسلة المتوفرة. في هذه الحالة، تقوم بإرجاع 3، والتي تم تخصيصها للمتغير `$length`. هذه هي الطريقة الأبسط والأكثر شيوعًا لاستخدام دالة.

المثال الثاني يمرر نتيجة `asin(1)` إلى دالة `sin()`. نظرًا لأن الدالتين الجيب والقوس هي انعكاسات، فإن أخذ جيب القوس لأي قيمة سيعيد دائمًا نفس القيمة. نرى هنا أنه يمكن استدعاء دالة داخل دالة أخرى. يتم إرسال القيمة المرجعة للإستدعاء الداخلي إلى الدالة الخارجية قبل إرجاع النتيجة الإجمالية وتخزينها في متغير `$result`

في المثال الأخير، نعطي اسم ملف لدالة `unlink()`، التي تحاول حذف الملف. مثل العديد من الدوال، فإنها ترجع false عندما تفشل. يتيح لك ذلك استخدام دالة مضمنة أخرى، وهي `die()` وخاصية قصر الدائرة "short-circuiting" لعوامل المنطق. وبالتالي، يمكن إعادة كتابة هذا المثال على النحو التالي:

```
$result = unlink("functions.txt") or die("Operation failed!");
```

على عكس المثالين الآخرين، تؤثر دالة `unlink()` على شيء خارج المعلمات المعطاة لها. في هذه الحالة، يقوم بحذف ملف من نظام الملفات. يجب توثيق كل هذه الآثار الجانبية للدالة بعناية ودراستها بعناية.

تحتوي PHP على مجموعة كبيرة من الدوال المحددة بالفعل لتستخدمها في برامجك. تؤدي هذه الملحقات مهام مثل الوصول إلى قواعد البيانات وإنشاء الرسومات وقراءة ملفات XML وكتابتها والاستيلاء على الملفات من الأنظمة البعيدة وغير ذلك.

ملاحظة:

لا تُرجع جميع الدوال قيمة. يمكنها تنفيذ إجراء مثل إرسال بريد إلكتروني ثم إعادة سلوك التحكم إلى استدعاء الكود؛ بعد إكمال مهمتها، ليس لديهم ما "يقولونه "say"".

تعريف الدالة

لتعريف دالة، استخدم الصيغة التالية:

```
function [&] function_name([parameter[, ...]])
{
 قائمة الجمل
}
```

يمكن أن تتضمن قائمة الجمل HTML. يمكنك تعريف دالة PHP لا تحتوي على أي كود PHP. على سبيل المثال، تعطي دالة ()column اسمًا قصيرًا مناسبًا لكود HTML التي قد تكون مطلوبة عدة مرات في جميع أنحاء الصفحة:

```
<?php function column()
{ ?>
 </td><td>
<?php }
```

يمكن أن يكون اسم الدالة أي سلسلة تبدأ بحرف أو شرطة سفلية متبوعة بصفر أو أكثر من الأحرف والشرطات السفلية والأرقام. أسماء الدالة غير حساسة لحالة الأحرف، وهذا يعني أنه يمكنك استدعاء دالة ()sin كـ sin(1), SIN(1), SiN(1) وهكذا، لأن كل هذه الأسماء تشير إلى نفس الدالة. حسب الاصطلاح، يتم استدعاء دوال PHP المدمجة بأحرف صغيرة.

عادةً ما ترجع الدالات بعض القيمة. لإرجاع قيمة من دالة، استخدم جملة return ضع return *expr* داخل دالتك. عند مصادفة جملة return أثناء التنفيذ، يعود عنصر التحكم إلى جملة الاستدعاء، وسيتم إرجاع النتائج المقيمة لـ expr كقيمة للدالة. يمكنك تضمين أي عدد من جمل الإرجاع في دالة (على سبيل المثال، إذا كان لديك جملة switch لتحديد أي من القيم العديدة التي تريد إرجاعها).

دعونا نلقي نظرة على دالة بسيطة. يأخذ المثال 3-1 سلسلتين، يسلسلهما، ثم يُرجع النتيجة (في هذه الحالة، أنشأنا مكافئًا أبطأ قليلاً لعامل التسلسل، لكننا نتحمل معنا من أجل المثال).

مثال 3-1. تسلسل السلسلة

```
function strcat($left, $right)
{
 $combinedString = $left . $right;

 return $combinedString;
}
```

<لم أنته بعد>